个性化你的阅读

编程狂人

Programming Madman

NO.6

推酷

关于推酷

推酷是专注于IT圈的个性化阅读社区。我们利用智能算法，从海量文章资讯 中挖掘出高质量的内容，并通过分析用户的阅读偏好，准实时推荐给你最感兴 趣的内容。我们推荐的内容包含科技、创业、设计、技术、营销等多方面内容， 满足你日常的专业阅读需要。我们针对IT人还做了个活动频道，它聚合了IT 圈最新最全的线上线下活动，使IT人能更方便地找到感兴趣的活动信息。

关于周刊

推酷周刊是专为IT人打造的行业技术周刊，目前推出的《编程狂人》是献给 广大的程序员们。 我们利用技术挖掘出那些高质量的文章，并通过人工加以筛 选整理出来。 每期的周刊一般会周一的某个时间点发布。

联系我们


tuicool2012


164644910

推酷网

下载APP

Android版本

iPhone版本

目录

业界新闻

前端开发

编程语言

VB 的未来计划

程序设计

C# 转 C++ 的一点分享

IOS 怎么用 UIScrollView 来滚动和缩放他的内容

Beanstalkd 队列 概述

Android 猜牌小游戏（改进版）

Android 开发心得—网页通过 webview 调用 Android 的图片或文件选择

后端架构

2013年个人微博推荐技术资料汇总

AliRedis 单机180w QPS, 8台服务器构建1000w QPS Cache 集群

文章： 聊聊并发（八）——Fork/Join 框架介绍

PostgreSQL 配置优化

数据库优化的最佳实践

分布式存储推荐论文

分布式文件系统 FastDFS 设计原理

程序人生

你是一个努力工作的程序员吗？还是一个懒惰的程序员？

各式各样的极品程序员，你属于哪一种

告别码农，成为真正的程序员

陋谈创业早期资金积累学习

新兴独立开发者的 PR 策略之语言与情感

正文

推荐 5 款 Google Analytics 的开源替代品

Web 分析软件通过收集、测量和分析网站访问者行为，向网站管理员和所有者提供实时数据，帮助他们识别机会和问题，跟踪流量来源，判断点击转换率等。Google Analytics 是搜索巨人提供的一个优秀 Web 分析数据服务，是最广泛使用的网站统计服务，能生成网站流量和来源详细统计数据。但 Google Analytics 的最大缺点是它被 Google 控制，可被 Google 用于其自身目的。Google Analytics 也不是一个开源解决方案，网站管理员无法访问到原始数据。其它基于云端的 Web 分析服务都存在类似的问题，如果你想在自己的服务器上托管一个开源的解决方案，这里也有许多优秀的替代，如支持48种语言的 Web 分析平台 Piwik，Open Web Analytics，AwStats 等等。

1. Piwik

Piwik 是一套基于 Php+MySQL 技术构建的开源网站访问统计系统，前身是 phpMyVisites。Piwik 可以给你详细的统计信息，比如网页 浏览人数， 访问最多的页面， 搜索引擎关键词等等，并且采用了大量的 AJAX/Flash 技术，使得在操作上更加便易。此外，它还采用了插件扩展及开放 API 架构，可以让开发人员根据 自己的实际需求创建更多的功能.

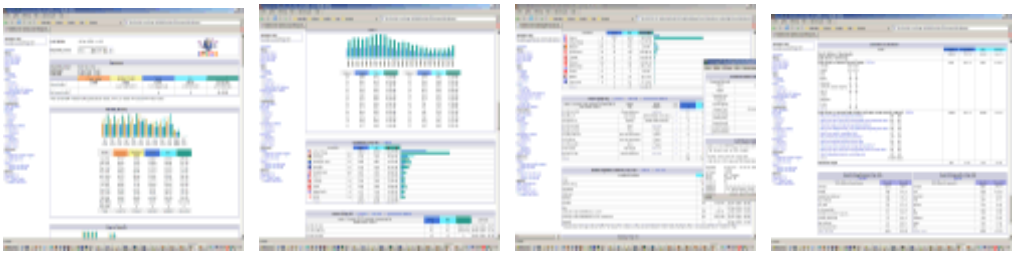
在线演示地址： http://piwik.org/demo/

2. Open Web Analytics

Open Web Analytics 是一个开源的网站流量统计系统。基于 PHP/Open Flash Chart/Ajax 技术开发，既可以单独使用也可以与 WordPress、Gallery&MediaWiki 集成使用。支持多个网站，集成 Google Maps，RSS/Atom 订阅跟踪等功能。

3. AWStats

AWStats 是一个免费的强大而有个性的工具,带来先进的网络,流量，FTP 或邮件服务器统计图. 本日志分析器作为 CGI 或从命令行在数个图形网页中显示你日志中包含的所有可能信息. 它利用一部分档案资料就能经常很快地处理大量日志档案,它能分析日志文件来自从各大服务器工具 ,如 Apache 日志档案 s (NCSA combined/XLF/ELF log format or common/CLF log format), WebStar, IIS (W3C 的日志格式)及许多其他网站,Proxy(代理服务器)、Wap、流量服务器、邮件服务器和一些 FTP 服务器 .

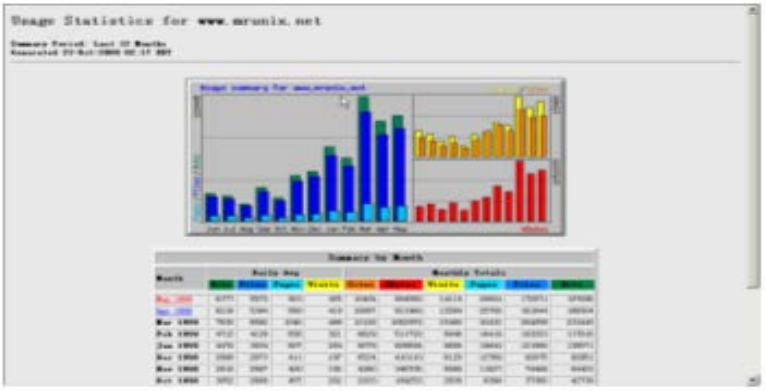


4. W3Perl

W3Perl 是一个 Web 日志的分析工具，支持 FTP、Squid、邮件日志等，提供一个图形化的界面，以及文本统计数据，提供一个管理界面。

5. webalizer

webalizer 是一个高效的、免费的 web 服务器日志分析程序。其分析结果是 HTML 文件格式，从而可以很方便的通过 web 服务器进行浏览。Internet 上的很多站点都使用 webalizer 进行 web 服务器日志分析。



原文

http://www.oschina.net/news/47230/google-analytics-opensource-alternative?utm_source=tuicool

Memcached 1.4.17 发布，集中式缓存系统

内存缓存 Memcached 1.4.17 发布。2013-12-21 上一个版本是 2012-12-09 的 1.4.16。此版本在 Linux 上支持了 accept4()，去掉每个 tcp 连接的 syscall，

增加 -F 参数可以禁止 flush_all 命令。 scripts/memcached-tool 支持 "settings" 和 "sizes" 命令及一些 Bug 修正。

完全改进：Fixes

```
    Fix potential segfault in incr/decr routine.
    Fix potential unbounded key prints (leading to crashes in logging code)
    Fix bug which allowed invalid SASL credentials to authenticate.
    Fix udp mode when listening on ipv6 addresses.
    Fix for incorrect length of initial value set via binary increment
protocol.
```

新特性：

```
    Add linux accept4() support. Removes one syscall for each new tcp
connection.
   scripts/memcached-tool gets "settings" and "sizes" commands.
    Add parameter (-F) to disable flush_all. Useful if you never want to
be
    able to run a full cache flush on production instances.
```

下载：http://www.memcached.org/files/memcached-1.4.17.tar.gz

原文

http://www.oschina.net/news/47147/memcached-1-4-17?utm_source=tuicool

evad3rs 公开信第二部分：已拒绝太极所有款项

12月25日，知名越狱团队 evad3rs 于几天前发布的 iOS 7越狱工具引起了轩然大波——当用户电脑语言为中文时，会自动安装名为“太极”的市场应用，而这个应用中存在着大量盗版软件。据传，为了捆绑在 Evasi0n 中，太极花费了“一百万美元”，甚至“数百万美元”。

evad3rs 于今天在越狱社区发布了公开信的第二部分，该部分对安全、盗版、以及收钱问题做了详细的解释。

亲爱的越狱社区，我们要对社会上一些额外的关注做出解释：

隐私与太极

首先要解释的是最重要，最令人关注的一点，是隐私。没有任何一个人数据被

泄露到任何地方。当然，作为越狱社区的成员，这次的事件有损我们7年来的安全声誉。但是我们需要重申的是，除非你的设备系统语言被设置成中文，否则不会有太极软件被安装。此外，除非你单独打开了太极的应用，否则也不会有太极软件运行。

之后有谣言四起，说我们有加密的数据被发送给了装了太极的中国用户。于是我们决定作了我们最擅长的——逆向研究了太极的代码用以了解有什么数据被发送了。太极的传输数据类似于 Cydia 的传输数据，唯一的设备识别码是以被加密的形式传送的，类似于 Cydia 中使用 SSL 来保护用户隐私的加密形式。太极完全没有传送用户的任何隐私数据到任何地方。

盗版与太极

我们的所有与太极的书面协议和口头协议均以取缔。虽然太极之前向我们保证没有盗版应用，但我们并没有去仔细检查他们商店的每一个应用安装包，我们当时只是做了一个粗略的检查，并没有发现任何问题。尽管如此，后来我们接到社区的一些调查和通知后才发现了一些盗版的应用。尽管起初我们并不相信太极是故意违反我们的协议，但这次事件对我们软件开发人员以及越狱社区的声誉的影响不可小觑，我们并不会在修复了越狱工具后就忘记这次事件的，我们会铭记于心并检讨。我们已经终止了与太极的合作关系。我们对太极居然让安装了太极的越狱工具破解版运行在他们网站上感到非常失望，我们并没有给他们任何许可或者源代码。

我们拒绝了所有来自太极的款项

目前社会上有很多传闻说我们收取了太极的款项。我们没有收取从任何团体收取任何款项，包括太极。我们以后也将不会接受任何金钱。 我们接受的捐款都已经给了公众机构，电子前沿基金会和自由信息基础设施基金会，以帮助保护越狱作为你的合法权利。

越狱的更新

我们正在努力解决越狱工具的各种问题。不幸的是，现在正是节日期间，我们的团队更想多花费时间陪伴我们的家人和朋友。所以我们现在的时间很紧张也颇有压力，我们需要多一些的时间来恢复。我们会竭尽全力努力工作以解决所有的遗留问题。感谢您对我们的理解。

我们非常努力地把越狱免费提供给所有人。我们也希望您可以享受我们的越狱工具。

原文 http://it.sohu.com/20131225/n392393914.shtml?utm_source=tuicool

pomelo 0.8 发布，网易游戏服务器框架

0.8 版本新特性

pomelo-cocos2d-jsb

对 cocos2d-x javaScript binding 环境有了支持，开发者可以很方便的使用 javaScript 来完成前后端的开发，并发布到 ios,android,web 平台上

详细请看 pomelo-cocos2d-jsb

pomelo 命令行重构

执行 pomelo 命令时，去除了对特定目录的依赖。

当在非 game-server 目录下执行 `pomelo start` 的时候，需要通过命令行选项指出代码的位置。对于 log4js 配置文件来说，需要将其日志文件的根目录设置为 game-server 下的目录。这个目录使用${opts:base}来指定，因此，如果在其他目录下启动应用服务器,请使用新的 log4js 配置,在 template 目录下有相应的示例,如下：

```
{


  "appenders": [


    {
      "type": "console"
    },
    {
      "type": "file",
      "filename": "${opts:base}/logs/con-log-${opts:serverId}.log",
```

```
    "pattern": "connector",
    "maxLogSize": 1048576,
    "layout": {
      "type": "basic"
    },
    "backups": 5,
    "category": "con-log"
  },

  {
    "type": "file",
    "filename": "${opts:base}/logs/crash.log",
    "maxLogSize": 1048576,
    "layout": {
      "type": "basic"
    },
    "backups": 5,
    "category":"crash-log"
  },

  // other appenders...
],

"levels": {
```

```
    "rpc-log" : "ERROR",
    "forward-log": "ERROR"
  },

  "replaceConsole": true,

  "lineDebug": false}
```

对于其他的 pomelo 子命令，如 `pomelo list`， `pomelo add`， `pomelo stop` 等 命令的执行也不再依赖目录，在执行时需要指定 master 服务器的地址及端口号，如果不指定，默认使用127.0.0.1:3005，因此这些命令可以远 程执行。当然执行这些命令的时候，需要相应的权限控制，默认的用户名和密码均为 admin，用户可以根据具体需求进行修改。

对于 `pomelo masterha` 命令，同 `pomelo start` 类似，也需要指出要执行代码的位置。这些修改保持与前面版本的兼容。

对 `pomelo start` 的改进

当以 daemon 方式启动 pomelo 应用的时候，去除了对第三方模块 forever 的依赖，此时启动的所有应用进程将脱离控制终端在后台运行，要查看所有的启动以及运行日志，只能通过相应的日志文件。

对于 `pomelo start` 的 env，可以支持由用户来自由配置，使用 -e|--env 选项。

使用 `pomelo --help` 可以获得更详细的 pomelo 命令行支持的选项。

pushScheduler 功能增强

对 response，push 增加了一个可选参数

在新版中，支持了更细粒度的 pushScheduler 选择。对 respone，push 增加了一个可选参数 userOption; 对 broadcast，扩展了原有的 option 参数的使用。用户可以把它定义为任意值，这个参数将会被用来选择具体的 pushScheduler 以及在 schedule 调用中使用，每一次的 response，push 以及 broadcast 都可以选择使用不同的 pushScheduler。具体示例如 下：

在 Handler 中，可以传入一个 userOption 参数，如下：

```
Handler.prototype.handle = function(msg, session, next) {

    // do handling

    // var resp = ...

    // var userOption = ...

    cb(null, resp, userOption);}
```

对于 push 操作，示例如下:

```
var aChannel = channelService.getChannel('aChannel');// var userOptions = ...aChannel.pushMessage(route, msg, userOptions, cb);
```

对于 broadcast 操作，以前的版本已经支持了 option 配置，具体的配置项仅仅支持了 binded 和 filterParam，新版本中对此进行了扩展，这两个选项继续有效，而且用户可以自定义更多新的选项，使用方式不变。

多个 pushScheduler 的配置及选择

在新版中，一个 connector 可以配置多个 pushScheduler，并根据自定义规则 selector 来进行选择，response 以及 push 新增加的可选参数 userOptions 可以用于 selector 的选择计算。下面的示例中就定义了三个不同的 pushScheduler，并应 用于不同的情况：

```
app.configure('connector-1', function() {


  app.set('pushScheduler', {


    scheduler: [
      {
        id: 'direct',
        scheduler: pomelo.pushSchedulers.direct      },


      {
```

```
      id: 'buffer5',
      scheduler: pomelo.pushSchedulers.buffer,
      options: {flushInterval: 5000}
    },

    {
      id: 'buffer10',
      scheduler: pomelo.pushSchedulers.buffer,
      options: {flushInterval: 10000}
    }
  ],

  selector: function(reqId, route, msg, recvs, opts, cb) {
    // opts.userOptions is passed by response/push/broadcast
    console.log('user options is: ', opts.userOptions);
    if(opts.type === 'push') {
      cb('buffer5');
      return;
    }
    if (opts.type === 'response') {
      cb('direct');
      return ;
    }
    if (opts.type === 'broadcast') {
```

```
        cb('buffer10');

        return ;

      }

   }

 });
```

新的 pushScheduler 配置与原有的一个 connector 仅仅支持一个 pushScheduler 的配置方式保持兼容。

rpc 调用方式改进

当进行 rpc 调用的时候，增加了跳过路由计算而直接将调用发送到一个具体的服务器或者广播到一类服务器的调用方式，代码实例如下：

```
// routeapp.rpc.<ServerType>.<Remote>.<Method>(routeParam, args..., cb);// to specified
server.app.rpc.<ServerType>.<Remote>.<Method>.toServer('server-id', args..., cb);// to all the servers whose type is
<ServerType>app.rpc.<ServerType>.<Remote>.<Method>.toServer('*', args..., cb);
```

生命周期回调

在 pomelo 0.8版中增加对外提供 application 的生命周期回调，这样能够让开发者在不同类型的服务器生命周期中进行具体操作。提供的生命周期回调函数包 括：beforeStartup，afterStartup，beforeShutdown，afterStartAll。其具体的功能说明如下：

beforeStartup(app, cb)

before application start components callback

Arguments

+ app - application object
+ cb - callback function

afterStartup(app, cb)

after application start components callback

Arguments

+ app - application object
+ cb - callback function

beforeShutdown(app, cb)

before application stop components callback

Arguments

+ app - application object
+ cb - callback function

afterStartAll(app)

after all applications started callback

Arguments

+ app - application object

具体使用方法：在 game-server/app/servers/某一类型服务器/ 目录下添加 lifecycle.js 文件，具体文件内容如下：

```
module.exports.beforeStartup = function(app, cb) {


    // do some operations before application start up

    cb();};module.exports.afterStartup = function(app, cb) {


    // do some operations after application start up

    cb();};module.exports.beforeShutdown = function(app, cb) {


    // do some operations before application shutdown down

    cb();};module.exports.afterStartAll = function(app) {
```

```
    // do some operations after all applications start up};
```

rpc filter 提供对外接口

根据网友的建议,在新版本中对外提供了添加 rpc filter 的接口,包括 rpcBefore、rpcAfter、rpcFilter，其功能分别为添加 before filter，添加 after filter，同时添加两种 filter；使用方法与 handler 的 filter 类似。在早期版本中提供的 enableRpcLog 选项，可以通过添加 rpc filter 代替。

```
app.configure('production|development', function() {



    // configurations



    app.filter(pomelo.filters.time());

    app.rpcFilter(pomelo.rpcFilters.rpcLog());});
```

简化配置文件

在0.8版的pomelo中对配置文件servers.json进行了精简,通过增加clusterCount字段将原有的配置进行了简化，这样将更加适合大规模应用的部署和运维管理。

原有的配置：

```
"connector":[

             {"id":"connector-server-1", "host":"127.0.0.1",
"port":4050, "clientPort": 3050, "frontend": true},

             {"id":"connector-server-2", "host":"127.0.0.1",
"port":4051, "clientPort": 3051, "frontend": true},

             {"id":"connector-server-3", "host":"127.0.0.1",
"port":4052, "clientPort": 3052, "frontend": true}

         ],

        "chat":[

             {"id":"chat-server-1", "host":"127.0.0.1", "port":6050},
```

```
            {"id":"chat-server-2", "host":"127.0.0.1", "port":6051},

            {"id":"chat-server-3", "host":"127.0.0.1", "port":6052}

        ],

        "gate":[

           {"id": "gate-server-1", "host": "127.0.0.1", "clientPort":
3014, "frontend": true}

        ]
```

0.8版本 pomelo 支持的简化配置：

```
"connector":[

            {"host":"127.0.0.1", "port":"4050++", "clientPort":
"3050++", "frontend": true, "clusterCount": 3}

         ],

        "chat":[

            {"host":"127.0.0.1", "port":"6050++", "clusterCount": 3}

        ],

        "gate":[

           {"host": "127.0.0.1", "clientPort": 3014, "frontend": true,
"clusterCount": 1}

        ]
```

同时在采用 pomelo-cli 进行动态增加服务器的时候，同样可以使用 add host=127.0.0.1 port=9000++ serverType=chat clusterCount=3 这样的形式同时增加多台服务器。

pomelo-logger 支持动态日志级别

在 pomelo-logger0.1.2中，增加了动态改变日志级别的功能，开发者可以在原有的 config/log4js.json 中配置 reloadSecs 参数，该参数表示定期检查配置文件是否有更新，如果有更新则重新装载并根据配置文件，更改日志级别。log4js.json 配置如下：

```
{


  ......


  "levels": {


    "pomelo" : "INFO",
    "rpc-log" : "INFO",
    "forward-log": "ERROR",
    "con-log": "INFO"
  },


  "replaceConsole": true,


  "reloadSecs": 60 * 3}
```

该配置表示每3分钟检查一次配置文件是否有更新。

安全性方面

RPC 调用的 IP 白名单

该功能可以为各个服务器的 RPC 调用提供 IP 白名单过滤功能.

原理

RPC 服务端每接受一个连接都会抛出一个连接事件, 这个事件中含有该连接的 socket.id 和 RPC 客户端 IP. RPC 服务端会捕获该连接事件, 并调用用户传入的获取 IP 白名单的函数, 如果该 RPC 客户端 IP 不在白名单中, 则立刻将对应的 socket 断开. 以此来实现 RPC 调用白名单过滤功能.

使用

使用时只需要向remoteConfig的配置中传入一个获取IP白名单的函数(whitelist: rpcWhitelist.whitelistFunc)即可，这个函数需要接受一个回调函数作为其参数，该回调函数形如function(err, tmpList) {...}. 在获取IP白名单的函数内，拿到IP白名单时(该白名单应为一维JS Array)，以类似于cb(null, self.gWhitelist)的形式调用IP过滤回调函数.

```
./game-server/app/util/whitelist.js... ...var self = this;self.gWhitelist = ['192.168.146.100', '192.168.146.101'];module.exports.whitelistFunc = function(cb) {

  cb(null, self.gWhitelist);

  };... ...

./game-server/app.jsvar rpcWhitelist = require('./app/util/whitelist');... ...// configure for globalapp.configure('production|development', function() {... ...

  // remote configures

  app.set('remoteConfig', {

    cacheMsg: true

    , interval: 30

    , whitelist: rpcWhitelist.whitelistFunc  });... ...}
```

具体请参考 lordofpomelo

pomelo-admin 的 IP 白名单

该功能可以为master服务器的admin提供IP白名单过滤功能.

原理

admin服务端每接受一个连接都会抛出一个连接事件，这个事件中含有该连接的socket.id和admin客户端IP. admin服务端会捕获该连接事件，并调用用户传入的获取IP白名单的函数，如果该admin客户端IP不在白名单中，则立刻将对应的socket断开. 以此来实现master服务器的admin白名单过滤功能.

使用

使用时只需要向masterConfig的配置中传入一个获取IP白名单的函数(whitelist: adminWhitelist.whitelistFunc)即可，这个函数需要接受一个回调函数作为其参数，该回调函数形如function(err, tmpList) {...}. 在获取IP白名单的函数内，拿到IP白名单时(该白名单应为一维JS Array)，以类似于cb(null, self.gWhitelist)的形式调用IP过滤回调函数.

```
./game-server/app/util/whitelist.js... ...var self = this;self.gWhitelist = ['192.168.146.100', '192.168.146.101'];module.exports.whitelistFunc = function(cb) {

  cb(null, self.gWhitelist);};... ...

./game-server/app.jsvar adminWhitelist = require('./app/util/whitelist');... ...// configure for globalapp.configure('production|development', function() {... ...

  app.set('masterConfig', {

    authUser: app.get('adminAuthUser') // auth client function

    , authServer: app.get('adminAuthServerMaster') // auth server function

    , whitelist: adminWhitelist.whitelistFunc  });... ...}
```

具体请参考 lordofpomelo

csv 配置文件自动热加载插件 pomelo-data-plugin

该插件可以监控指定目录下的所有 csv 格式的配置文件，并在某个文件被改变时自动地将其热加载进入 Pomelo.

原理

该插件主要使用 csv 模块来解析 csv 配置文件，使用 `fs.watchFile` 函数来监控文件变化事件. 当 Pomelo 框架启动该插件中的组件时，组件会加载给定文件夹中的所有 csv 配置文件，并为每个文件加一个 watcher. 以此来实现 csv 配置文件自动热加载的功能.

安装

```
npm install pomelo-data-plugin
```

使用

```
./pomelo-data-plugin-demo/config/data/team.csv# 队伍名称配置表# 队名编号，队名 id,teamName5,The Lord of the Rings6,The Fast and the Furious
```

上面的 csv 配置文件中，以#开头的行是注释语句，正文的第一行应为列名(`id` 为该文件的主键列名，为必须列)，下面的行是对应列名的具体数据.

```
var dataPlugin =
```

```
require('pomelo-data-plugin');... ...app.configure('production|development', function() {

  ...

  app.use(dataPlugin, {

    watcher: {

      dir: __dirname + '/config/data',

      idx: 'id',

      interval: 3000

    }

  });

  ...});... ...... ...var npcTalkConf = app.get('dataService').get('npc_talk');... ...... ...
```

上面代码中的 `dir` 即为需要监控的配置文件夹; `idx` 为所有 `csv` 配置文件的主键列名(如:`team.csv` 所示的 `id`); `interval` 为 `fs.watchFile` 函数测试其所监控文件改变的时间间隔, 单位为毫秒.

原文 http://www.oschina.net/news/47195/pomelo-0-8?utm_source=tuicool

我国自主操作系统 SpaceOS 具备同时运行 Windows 和 Linux 应用软件的能力

SpaceOS 是什么？通过下面简短来了解 SpaceOS，SpaceOS 最重要，也是大家最关心的能同时运行 Windows 和 Linux 应用软件的能力究竟如何？对不起，这个由于没有面世，外人不得而知。

QUOTE:

Space os 操作系统是我国自主知识产权的计算机操作系统，是中国科学院计算机联合研究院的刘金刚教授花费10年的时间研制出来。具有更安全，更快速，具有强大兼容性的系统，具备同时运行 Windows 和 Linux 应用软件，支持三维桌面系统，三维显示效果等特点，是具有多项国家专利权的电脑操作系统。

Space OS 电脑桌面系统是目前唯一成功实现跨平台操作的系统基础软件，可以看出 Space OS 电脑最终实现一个新的架构体系。

全新的体验、安全可靠、软硬件一体、解除一切烦恼。新颖的安全模式、跨平台使用功能、升级简单、快捷操作方便、易行。专用于军队、政府机关、公安、金融、财务、学校、企事业、工控等需要安全可靠的系统和部门。

Space OS 电脑的操作方便快捷，拥有动态视窗，文件处理，三维桌面系统，三维显示效果等功能。

[图]SpaceOS 应用的领域令人吃惊。

原文 http://www.linuxsight.com/blog/7534?utm_source=tuicool

圣诞节快乐：来自程序员们的问候

摘要：转眼间，一年一度的圣诞节又来临了。在这个越来越受到国人重视的节日中，每个人有每个人的浪漫方式，当然程序员们也不例外。来看看程序员们是如何为这个节日增添不一样的气氛的。

转眼间，一年一度的圣诞节又来临了。在这个越来越受到国人重视的节日中，每个人有每个人的浪漫方式，当然程序员们也不例外。

下面就来看看一些 IT 企业和程序员们是如何为这个节日增添不一样的气氛的。

一、来自 IT 巨头的问候

1. Google 的彩蛋

每到节日，Google 就会在其搜索引擎中加入一些彩蛋，在用户搜索特定关键词的时候出现特殊的页面效果。临近圣诞，当用户在 Google 中输入“圣诞节”时，会页面上端出现飘雪的场景和圣诞老人驾驶鹿车飞驰的画面。

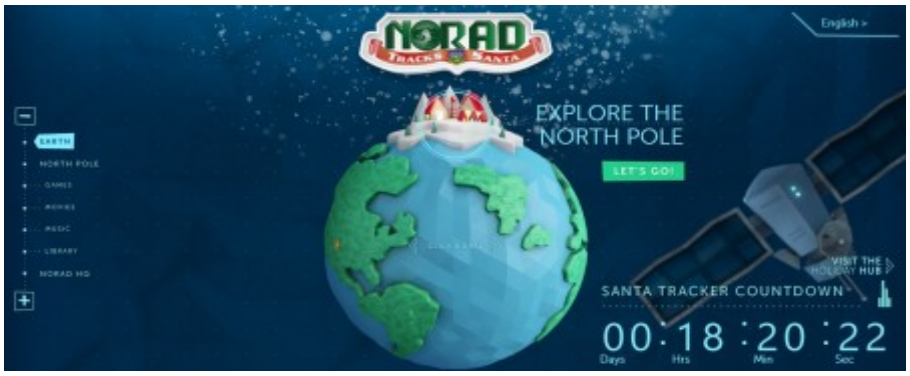
2. 百度的小游戏

当用户在百度中搜索“圣诞节”时，就会出现一个小游戏。该游戏基于 HTML5制作。

详细信息可查看这个页面的源码。
http://www.baidu.com/ur/show/uhchristmas?from=christmasresult

3. 微软追踪圣诞老人网站

微软今年推出了一个追踪圣诞老人的网站，该主题网站基于 HTML5和 WebGL 技术，首页有圣诞节的倒计时，此外还有许多好玩的在线游戏，最主要的是该网站在触摸设备上也有很好的体验。



网站地址：http://www.noradsanta.org/

二、来自初级程序员的问候

如果你刚开始学习 HTML5，你可以制作一些简单的效果来庆祝圣诞节。

1. 使用 canvas 功能绘制的简单圣诞树

在绘图板中绘制下面这个图形并不算什么难事，但是使用代码来生成这个圣诞树却需要一定的 HTML5基础。

下面这个图形使用 HTML5中的<canvas>标签来绘制，如果你熟悉 HTML5，这对你来说轻而易举。

源码：http://www.spjeff.com/2013/12/05/christmas-tree-html5-js-and-css3/

2. 雪花效果

jQuery 的出现，让各种动画效果变得更加容易。比如，你可以通过 jQuery、jQuery.snow.js 插件以及少量的代码，就可以让页面中飘舞这雪花。

源码：https://github.com/tzach/merry-christmas

3. 一个非常漂亮的圣诞贺卡

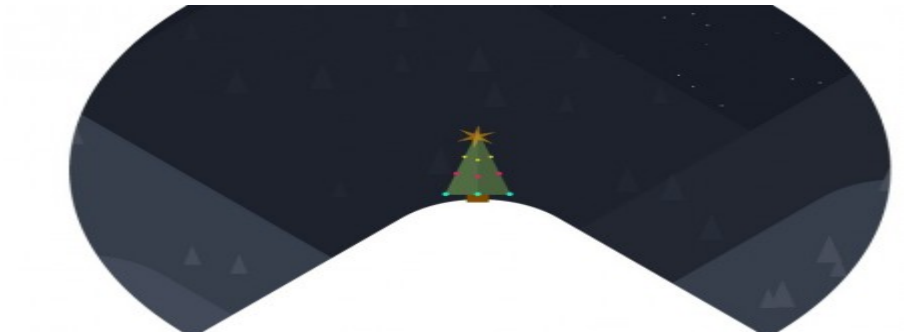
该贺卡通过 Construct2制作，然后通过 c2runtime.js 使得该贺卡可以直接在网页中运行。Construct2是一款用来制作 HTML5应用的软件，拥有一个清晰直观、支持“拖拽”操作的开发环境，即使你没有任何编程经验也能开发自己的 HTML5应用。

演示：http://3.s3.envato.com/files/72733144/blue/index.html

三、来自中级程序员的问候

随着编程技能的进一步掌握，你可以使用稍复杂的技术来实现一些更绚丽的效果。

1. **CSS3实现圣诞树动画**该动画主要使用 CSS3实现各种转场效果，并使用了 HTML5 中的<audio>标签来播放音频文件。同时还使用了 StyleFix 和 PrefixFree 脚本，这样在编写代码时可以不用为特定的 CSS3属性添加浏览器前缀，也可以在其他浏览器中播放。



演示：http://christmasexperiments.com/2013/11/experiment.html

源码：https://github.com/podrivo/christmas

2. HTML 表单元素制作的圣诞树

下面这个圣诞树没有采用松树的形式，而是采用了 HTML 表单元素来制作，比如输入框、单选钮，进度条，按钮等。

演示：http://hakim.se/experiments/css/domtree/

源码：https://github.com/hakimel/DOM-Tree

3. 使用 tree.js 库创建圣诞树

three.js 是一款开源的 JavaScript 3D 框架，也可以说是一款 WebGL 框架，几乎可以实现所有的3D 场景。本文所提到的圣诞树就是用 three.js 和 HTML5技术实现的。

演示：http://www.script-tutorials.com/demos/372/index.html

源码：http://www.script-tutorials.com/christmas-tree-with-threejs/

three.js 托管地址：https://github.com/mrdoob/three.js/

4. 3D 雪花效果

该效果使用 HTML5的<canvas>标签和 three.js 实现了3D 的雪花飞舞效果。你可以拖动鼠标进行旋转。

演示：http://seb.ly/demos/JSSnowNew/snow3d.html

源码：
https://github.com/sebleedelisle/live-coding-presentations/tree/master/2011/JSSnow

四、来自高级程序员的问候

1. 代码不到1KB 的3D 圣诞树

下面这个3D 圣诞树只用1021字节的 JavaScript 代码编写而成，逼真的3D 及旋转效果，将 JavaScript 功能发挥到了极致。

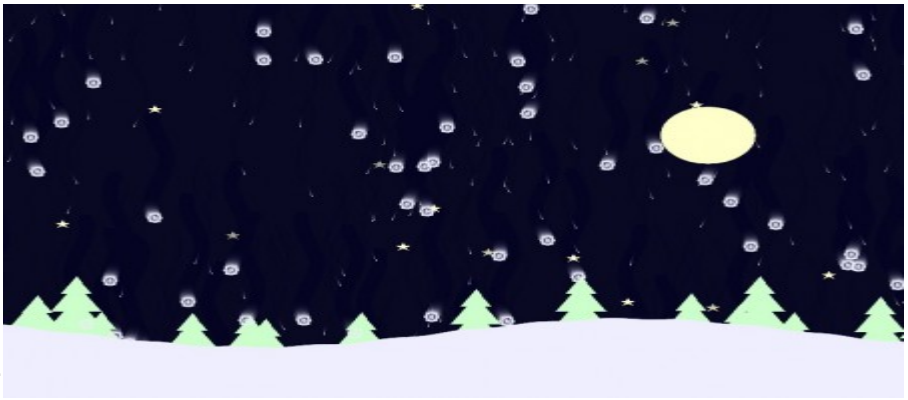
演示：http://js1k.com/2010-xmas/demo/856

源码如下：

[js] view plaincopy

```
M=Math;Q=M.random;J=[];U=16;T=M.sin;E=M.sqrt;for(O=k=0;x=z=j=i=k<200;)with(M[k]=k?c.cloneNode(0):c){width=height=k?32:W=446;with(getContext('2d'))if(k>10|!k)for(font='60px Impact',V='rgba(';I=i*U,fillStyle=k?k==13?V+'205,205,215,.15)':V+(147+I)+','+(k%2?128+I:0)+','+I+',.5)':'#cca',i<7;)beginPath(fill(arc(U-i/3,24-i/2,k==13?4-(i++)/2:8-i++,0,M.PI*2,1)));else for(;x=T(i),y=Q()*2-1,D=x*x+y*y,B=E(D-x/.9-1.5*y+1),R=67*(B+1)*(L=k/9+.8)>>1,i++<W;)if(D<1)beginPath(strokeStyle=V+R+','+(R+B*L>>0)+',40,.1)'),moveTo(U+x*8,U+y*8),lineTo(U+x*U,U+y*U),stroke();for(y=H=k+E(k++)*25,R=Q()*W;P=3,j<H;)J[O++]=[x+=T(R)*P+Q()*6-3,y+=Q()*U-8,z+=T(R-11)*P+Q()*6-3,j/H*20+((j+=U)>H&Q()>.8?Q(P=9)*4:0)>>1]}setInterval(function G(m,l){A=T(D-11);if(l)return(m[2]-l[2])*A+(l[0]-m[0])*T(D);a.clearRect(0,0,W,W);J.sort(G);for(i=0;L=J[i++];a.drawImage(M[L[3]+1],207+L[0]*A+L[2]*T(D)>>0,L[1]>>1)){if(i==2e3)a.fillText('Merry Christmas!',U,345);if(!(i%7))a.drawImage(M[13],((157*(i*i)+T(D*5+i*i)*5)%W)>>0,((113*i+(D*i)/60)%(290+i/99))>>0);}D+=.02},1)
```

2. **代码不到1KB 的雪景效果**



下

面这个场景也是由不到1KB 的 JavaScript 代码生成。

演示：http://js1k.com/2010-xmas/demo/855

源码如下：

[js] view plaincopy

```
for(p in a)a[p[0]+(p[6]||'')]=a[p];var
M=Math,C=M.cos,S=M.sin,R=M.random,T=0,x=[],y=[],W=innerWidth,H=inner
Height,L=2047,Z=100,V=20,N=511,M=1337;c.width=W,b.style.overflow='hi
dden',b.style.margin='0px',c.height=H;g='globalAlpha';h='fillStyle';
for(i=N;i--;){x[i]=L*R();y[i]=L*R()}setInterval(function(){T+=1/V;a[
g]=0.2;function
m(c){a[h]=c}m('#002');a.fc(0,0,W,H);a[g]=1;u=1;m('#ffc');for(i=0;i<N
/3;++i){d=u=(u*M+1)&L;u=(u*M+1)&L;if(C(T*Z+i)<0.5){a.fx("\u2605",d,u
)}}a.ba();a.arc(2*W/3,H/3,40,0,6.3,1);a.ca();a.fill();m('#cfc');B=H-
V;for(j=0;j<5;++j){s=90-j*V;a.font=s+"px
serif";F=1;for(i=0;i<W;i+=(F=(F*M)%Z)){a.fx("\u25B2",i-s/2,B+S(i)*30
);}B-=s/2;}m('#eef');for(i=N;i--;){e=x[i];f=y[i];a.fx("\u06DE",e,f);
y[i]=(f+1)%L;x[i]=(e+C(i+T)/3)%L}for(i=N*3;i--;){a.fc((x[i&N]+i)&L,(
y[i&N]+i)&L,1,1)}for(i=W;i--;){d
=Z+V*S(i/Z)+S(i/10);a.fc(i,H-d,1,d)}},50);
```

此外还有很多1KB 代码编写的圣诞效果，大家可以访问 js1k 圣诞主题页面。

3. 游戏般的3D 雪地场景

整个 Demo 是基于一个无限开阔的雪地场景的，里面有圣诞树和雪人，可以像玩 FPS 游戏一样在里面走动，WASD 操控移动，按住鼠标左键拖拽控制方向，整个 Demo 是使用 Oak3D 框架制作的。

演示地址：http://christmas.oak3d.com/Scene/MerryChristmas.html

五、自己动手

看完上面的这些效果，你是不是也想自己动手做一个属于自己的圣诞礼物呢。下面我们为你准备了一些设计素材和教程。

1. 80套圣诞节主题设计素材
2. 16个令人印象深刻的 HTML5/CSS3/Javascript 实验
3. 49个免费的圣诞主题矢量图像
4. 40个最好的圣诞主题资源
5. 其他圣诞节设计资源
6. 来自 w3school 的 HTML5教程

原文
http://www.csdn.net/article/2013-12-23/2817894-Merry-Christmas?utm_source=tuicool

写给设计师看的 HTML&CSS 入门指导

想做一个开发&设计并行的牛人？却不知道如何开始？欢迎观看本文，说不定会引起你对开发的兴趣
HTML&CSS 是网页设计的基本。设计师应该时刻学习，不要仅仅局限于视觉设计的层面。那么该要如何开始 HTML&CSS 的学习呢？
本文勾勒出了 HTML&CSS 的基本，即便不学，掌握这些基本知识也是很有必要的。

整体简介

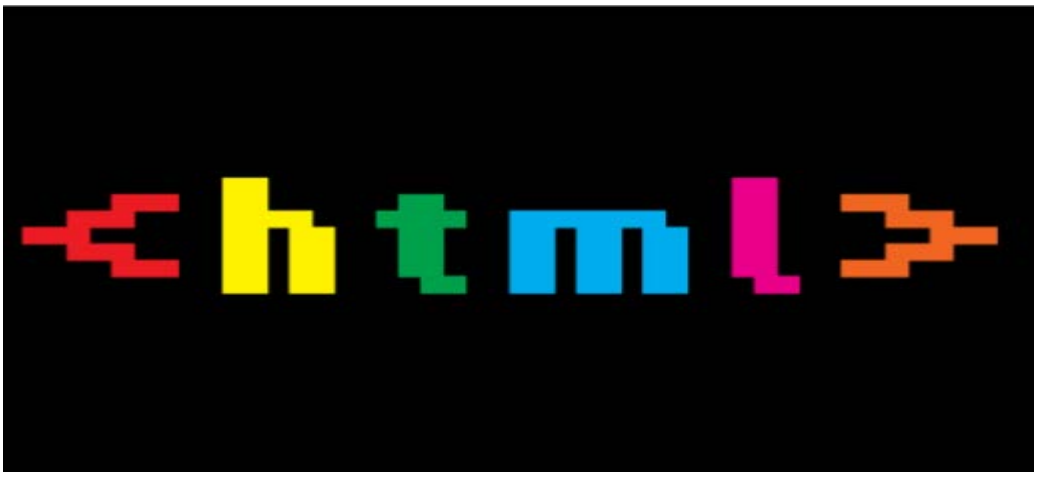
在开始学习 HTML&CSS 之前，首先要搞清楚两者的区别。两者在整体上有着很明显的差异。

整体看来，HTML 是超文本标记语言，是用来构建框架的。

而 CSS 值得是层叠样式表，统一控制 HTML 网页的外观属性。

（而 JavaScript 是行为）

HTML 语言的基本



标签、属性、元素是在掌握之前必须要了解的几点基础。

元素
元素是用来定义页面、内容、结构的一种标识符。一些流行的元素包括 h1-h6，p，div，a，span，strong 以及 em.

案例：

< a >

标签
标签主要有两种形式，开始标签和结束标签(也称为开放标签和闭合标签)。

HTML 元素以开始标签起始
HTML 元素以结束标签终止

案例:

<a>……

HTML 链接由 <a> 标签定义。链接的地址在 href 属性中指定：This is a link

…..

HTML 段落是通过 标签进行定义的。 This is a paragraph.

属性

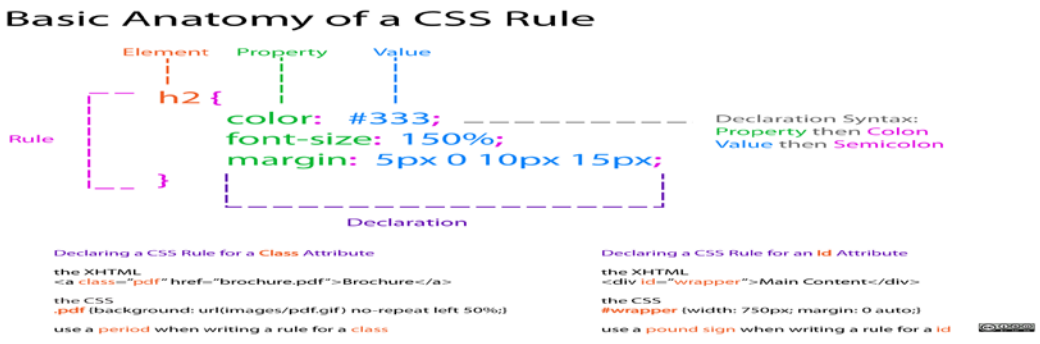
属性指的是赋予元素的附加信息。大多数情况下，属性将标题、类、ID 分配格元素，用来描述媒体元素的来源，并可作为超链接的参考。

案例：

Example

HTML 链接由 <a> 标签定义。链接的地址在 href 属性中指定：

CSS 语言的基本



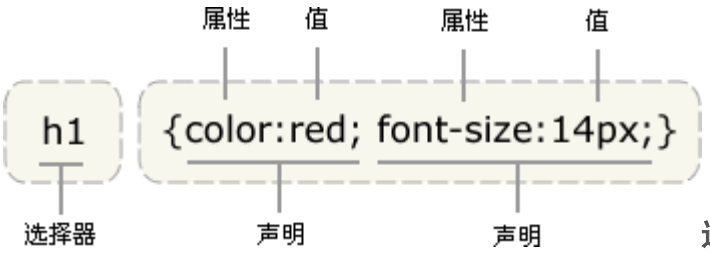
除了 HTML，你还要了解 CSS，这是你第二件必备武器。下面这个案例展示了 CSS 的几个基本

案例:

在这个例子中，h1 是选择器，color 和 font-size 是属性，red 和 14px 是值。

```
h1 {color:red; font-size:14px;}
```

下面的示意图为您展示了上面这段代码的结构：



选择器能够为确定元素加入指定样式，包括 ID、类、标签选择器等等。

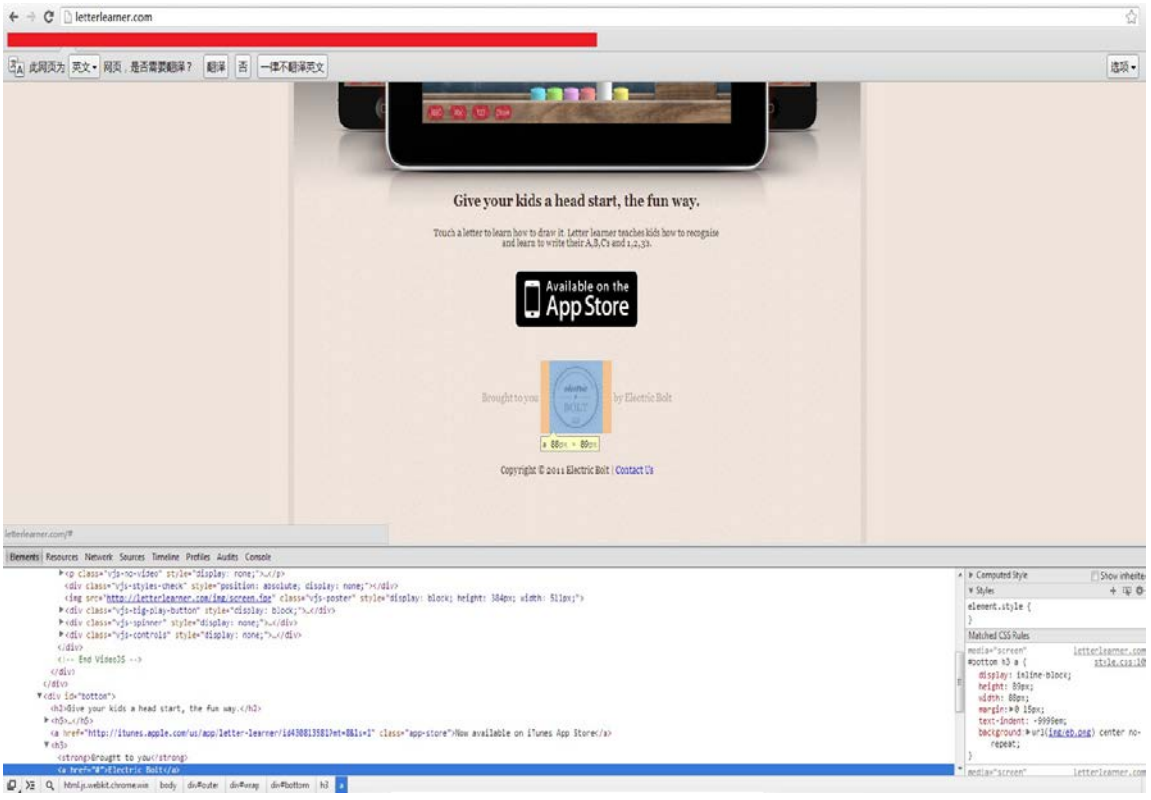
属性

属性定义了为元素加入样式的种类，诸如颜色、字体大小等等。（冒号之前）

值

样式种类的具体数值。（冒号和分号之间）

框模型和定位



另外需要了解的便是框模型。CSS 框模型 (Box Model) 规定了元素框处理元素内容、内边距、边框 和 外边距 的方式。

关键便是 margin 和 padding 属性，而正确理解这两个属性也是学习用 css 布局的关键。

CSS 框模型

border(边框)

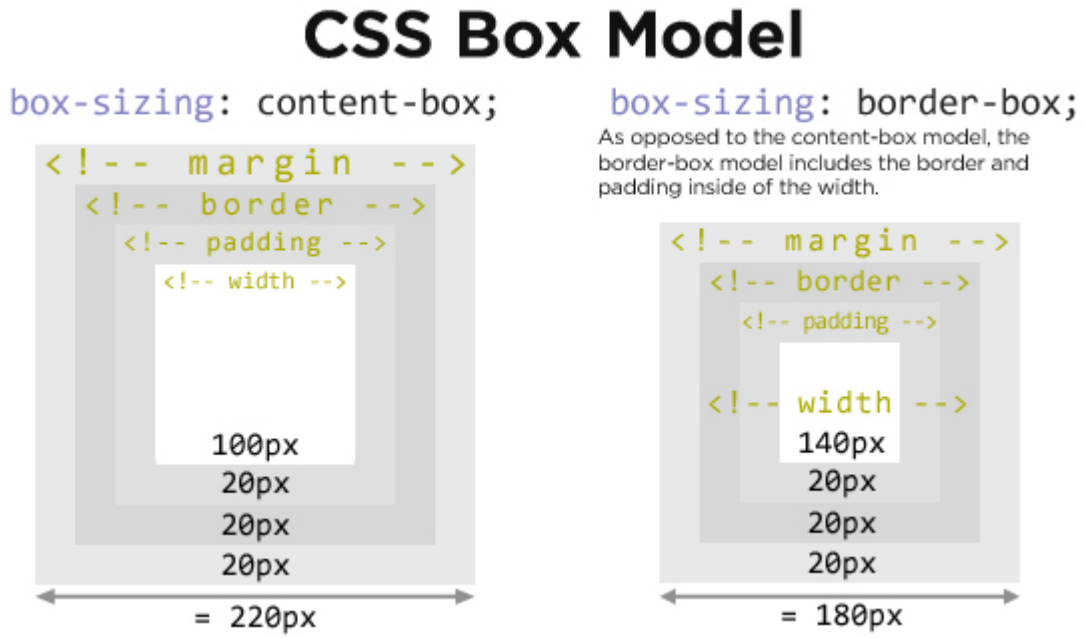
margin(外边距)

padding(内边距)

width(宽度)

元素框的最内部分是实际的内容，直接包围内容的是内边距。内边距呈现了元素的背景。内边距的边缘是边框。边框以外是外边距，外边距默认是透明的，因此不会遮挡其后的任何元素。

框模型包括以上几种,可调整大小,框模型便能以这种方式影响内容传递的方式。

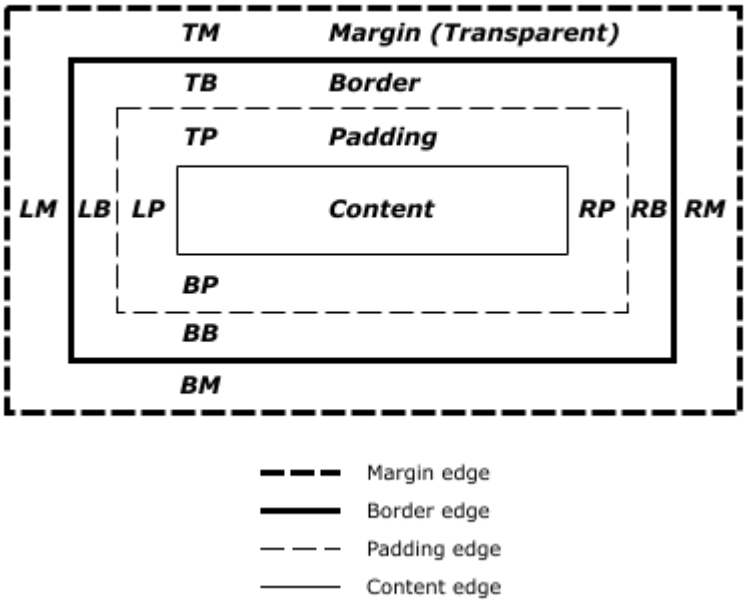


HTML 框模型

元素的高度和宽度可自行设定,然而整体内容的高度和宽度受内边距以及边框影响。

案例:

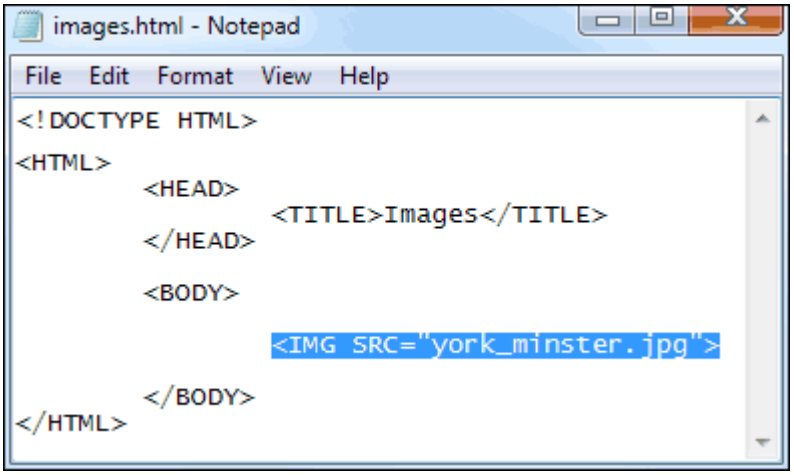
```
div {

background: #fff;

border: 6px solid #ccc;

height: 100 px;

margin: 20 px;

padding: 20px;

width: 400px;

}
```



添加图像、音频、视频

文本与多媒体结合能达到更好的传递效果。

添加图像
图中内含的文本部分便能实现添加图像功能，



不可忽视的两个属性：

src 属性：src 属性可设置或返回应当被载入框架中的文档的 URL。

alt 属性：alt 属性是一个必需的属性，它规定在图像无法显示时的替代文本。

实例：

```
<img src="/i/eg_tulip.jpg"  alt="上海鲜花港 - 郁金香" />
```

亲自试一试

如果无法显示图像，浏览器将显示替代文本，就像这样：

上海鲜花港 - 郁金香

此外，当用户把鼠标移到图像上方，最新的浏览器会在一个文本框中显示描述性文本。下面的代码在 alt 属性中为图像添加了描述性文本：

您可以把鼠标移动到下面的照片上，看看相应的效果：

红圈中的文本框是用户把鼠标移动到图像上时，IE7 浏览器所显示的效果：

添加音频

HTML5提供了一种简易的方式添加音频和视频文件。跟 img 元素一样，音频元素也需要 src 属性来提供 URL 来源


```
<HTML>
<HEAD>
<TITLE>Sound Example 1: Bgsound Tag</TITLE>
<SCRIPT LANGUAGE="JavaScript">
<!--
function PracticeWindow() {
window.open("try.htm","MainWindow","status,resizable,height=538,width=4
}
//-->
</SCRIPT>
</HEAD>
<BODY BGCOLOR="#FFFFFF" TEXT="#000000" LINK="#0000FF" VLINK="#cc3299" A
<BGSOUND SRC="waitng2.wav" LOOP="2">
<CENTER>
<B>
<A NAME="TOC"><H1>Bgsound Tag Example</H1></A>
</B>
</CENTER>
<HR SIZE=3>
```

添加视频
跟添加音频同理，只不过用视频元素来取代音频元素。

```
<object width="320" height="320"><param
value="http://popplet.com/app/Popplet_Alpha.swf?page_id=580600&e
m=1" name="movie"></param><param value="true"
name="allowFullScreen"></param><param value="always"
name="allowscriptaccess"></param><embed
src="http://popplet.com/app/Popplet_Alpha.swf?page_id=580600&em=
1" height="320" width="320" allowfullscreen="false"
allowscriptaccess="always" type="application/x-shockwave-
flash"></embed></object>
```

希望这份入门指导能让你对 HTML&CSS 产生兴趣，同时教会你 HTML&CSS 的基本，其实它们学起来很简单。

原文 http://www.uisdc.com/html-and-css-guide?utm_source=tuicool

CSS 忍者：安全的 CSS hacks 秘籍

你如何搞定 IE 这只难以驯服的怪兽？使用 CSS Hacks 或者条件注释么？恐怕没有什么解决方案是完美的。每个设计师或者前端码农都会有自己打败 IE 行之有效

的方法。所有这些技术都各有利弊，让我们一起来看看。

通过条件判断引入样式表

使我们能很容易在 IE 浏览器中通过条件注释语句加载指定样式表，而其他非 IE 内核浏览器则自动忽略这段蹩脚的 HTML 注释。下面是一个例子：

```
<!--[if IE 8]><link rel="stylesheet" href="ie8.css"><![endif]-->

<!--[if IE 7]><link rel="stylesheet" href="ie7.css"><![endif]-->

<!--[if IE 6]><link rel="stylesheet" href="ie6.css"><![endif]-->
```

这段代码会导致 IE8、IE7、IE6 各自加载对应的样式文件。这事实上是非常牛逼的，条件注释给按浏览器加载各自不同的样式表提供了可能，但同时需要维护多个样式文件。

CSS hacks

这事实上是个龌龊的做法，能解决问题又不符合规范，看着也很头大。大部分人看着它只能束手无策而又逼不得已。之前的《CSS Hacks for IE，IE 也可以很完美》已经谈过 IE 的 CSS hacks 了。现在可以来简单回顾一下常用的几个方法：

```
_selector{property:value;} //IE6

*selector{property:value;} //IE6 & IE7

selector{property:value\9;} //IE6 & IE7 & IE8
```

这个一般人都知道，就不多说了。但面临的一个现实问题是，\9 这个 hack 不只是针对 IE8 以及更老版本奏效，IE9 最终的发行版依旧会受到这个 hack 的影响。而 IE9 本身在 CSS 上的各种缺陷又是被修复的。如果将来 IE10、IE11 来了，那又怎么办？显然 \9 并不是一个严谨的安全的方案。

另外，采用不同的 X-UA-Compatible 模式也会影响不同版本的 IE 渲染差异。建议设置默认渲染模式如下：

```
<meta http-equiv="X-UA-Compatible" content="IE=Edge">//标准 IE 模式
```

那咋整呢？咋整咋整咋整？没事，下面还有更绝的。

通过条件判断插入指定类

既然可以用条件判断，那么直接为不同 IE 版本输出单独用于设定样式的钩子类好

了。直接上代码：

```
<!--[if !IE]><html><![endif]--> // 非 IE 浏览器的情况，不添加任何作用类

<!--[if IE 6]><html><![endif]-->

<!--[if IE 7]><html><![endif]-->

<!--[if IE 8]><html><![endif]-->
```

如果要是 IE9、IE10 再想出现什么神奇的行为艺术的话，继续添加指定作用类就行咯。并且你的样式表也会变得异常干净、整洁、美观：

```
.selector { color: black; }

.ie8 .selector { color: green; } /* IE8 */

.ie7 .selector { color: blue; } /* IE7 */

.ie6 .selector { color: red; } /* IE6 */
```

当然，标准模式的 X-UA-Compatible 声明还是必须的，以防页面被强制按照低版本的 IE 来渲染。

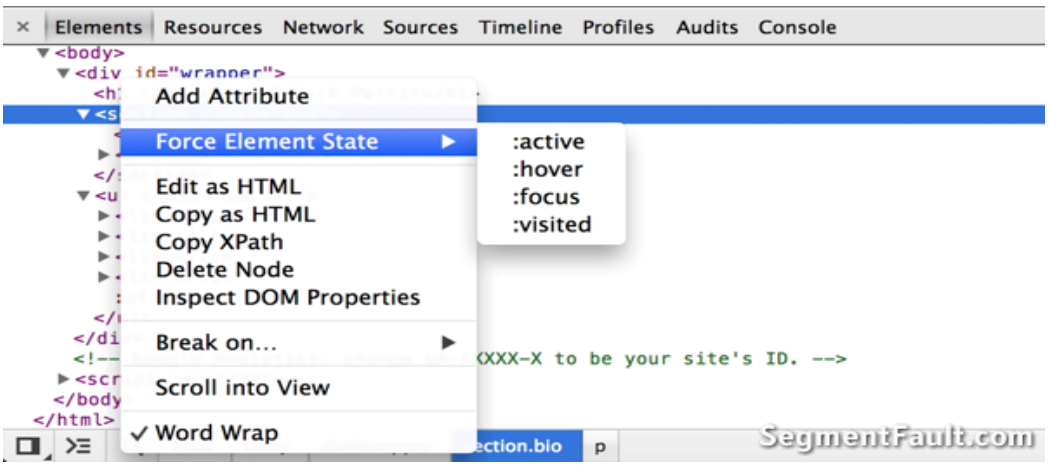
原文 http://blogread.cn/it/article/3813?f=hot1&utm_source=tuicool

四个实用但容易忽略的 Chrome 开发工具特性

Chrome 开发工具是基于 Chrome 浏览器，帮助开发人员调试代码的控制面板。它的功能很丰富以至于我们很难面面俱到，这里就有几个不太明显但很实用的功能。

改变 DOM 元素状态

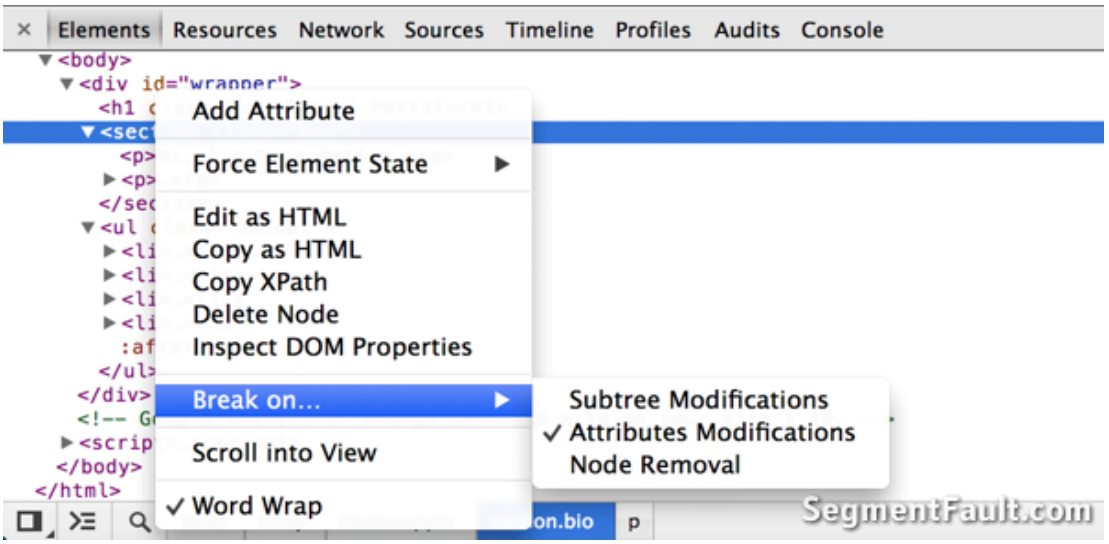
右击 DOM 元素，会列出功能选项，其中有一项名为'Force Element State'，展开子菜单可以看到几种常见的伪类：`:active`, `:hover`, `:focus`, and `:visited`。



如果在你的 CSS 或 JS 中想要调试这些伪类效果，就可以使用这个功能。

DOM 断点

我们熟悉 JS 的断点调试，但很少有人知道 DOM 也可以断点调试！依然右击 DOM 元素，可以看到一个名为 **Break on** 的选项，展开有 `Subtree Modifications`，`Attributes Modifications` 以及 `Node Removal` 三个选项。



当 JS 尝试改变 DOM 元素时，给元素添加的断点便会触发。

- Subtree Modifications，当添加，改变，删除子元素时触发
- Attributes Modifications，当元素属性被改变时触发
- Node Removal，当移除元素时触发

Console API

我们最为熟悉的一定是 `console.log`，但 console 的方法远远不止这一个，有兴趣的可以看看官方文档。这里再讲一个很实用的方法——`console.count`，它可以统计代码块执行的次数，如果你怀疑一个函数是不是被执行了很多次，用它就可以很轻松的统计出来。

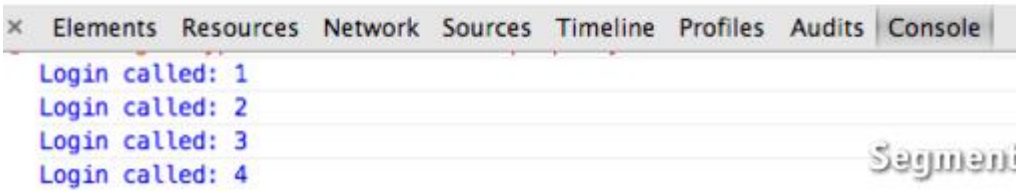
```
function login(user) {
```

```
    console.count("Login called");

    // login() code...

}
```

这样，每次调用 `login` 时都会在控制台打印出次数

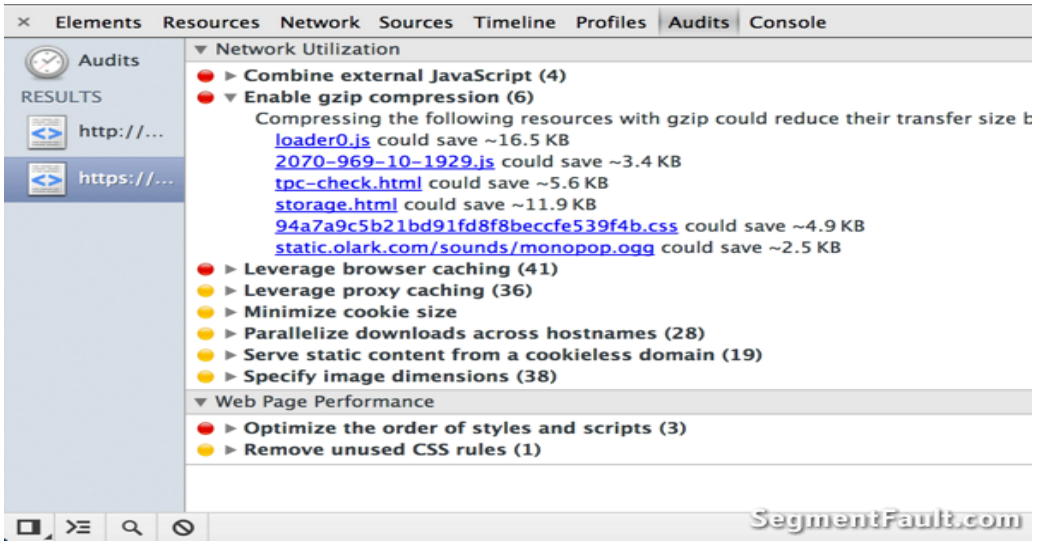
```
function login(user) {
    console.count("Login called");
    // login() code...
}
```

这样，每次调用login时都会在控制台打印出次数



Audits

Audits 可以检查页面的性能方面存在的问题，并给出优化意见，比如 CSS 和 JS 的引入位置，是否开启了 Gzip 压缩，是否很好地利用了浏览器缓存以及没有应用到的 CSS 规则等，是不是很智能很强大呢？



原文

http://blog.segmentfault.com/laopopo/1190000000370378?utm_source=tuicool

CSS 选择器详解（二）通用选择器和高级选择器

通用选择器

通用选择器可以选择页面上的所有元素，并对它们应用样式，用 * 来表示。

语法：

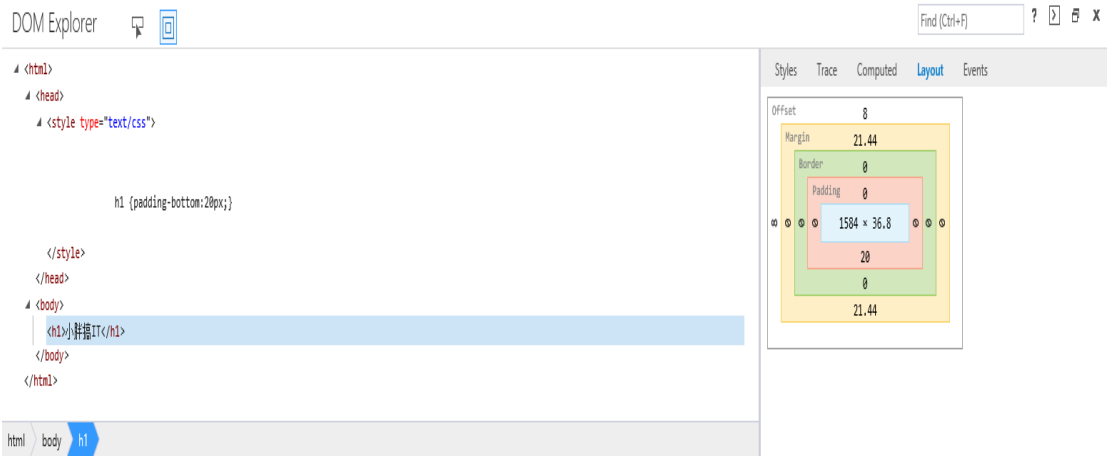
```
* { property1: value; property2: value; }
```

示例：

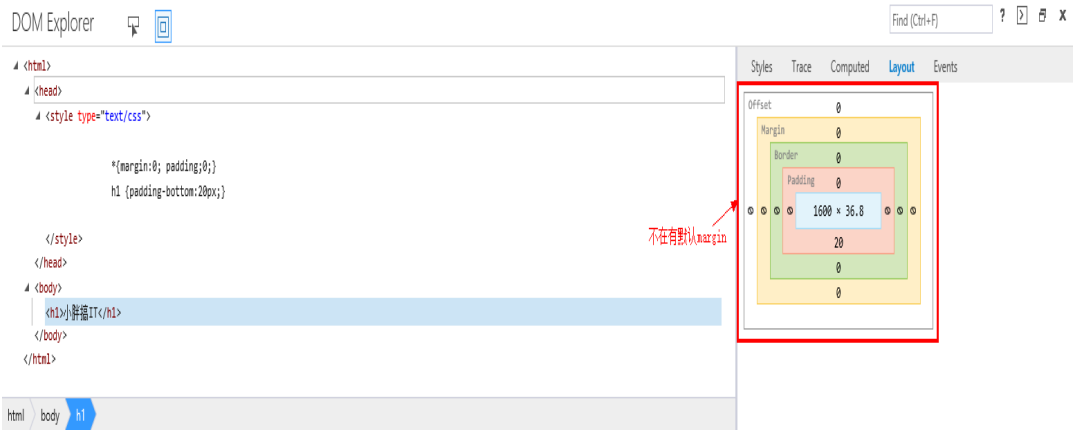
```
* { margin:0; padding:0;}
```

这行代码可以删除每个元素在浏览器中 margin 和 padding 的默认值。不同的浏览器对元素的默认 margin 和 padding 可能不同，用通用选择器把所有元素的 margin 和 padding 都设置为0方便我们精确地控制元素的 margin 和 padding。

此处我们以 IE11为例看一下实际效果：



当我们想要 h1距离下一个元素的距离为20（此处假设下一个元素的 margin-top, padding-top 和 border-weight 都是0），可以通过设置 padding-bottom 为20px 来实现，但观察效果却发现 h1距离下一个元素远不止20px，这是由于 IE11对 h1有一个默认的 margin 值（可以观察到其实 body 也是有默认 margin 的），可以通过通用选择器来修复这个问题：



子选择器

后代选择器选择一个元素的所有后代，而子选择器只选择元素的直接后来，即后代的后代不会受影响。

语法：

```
selector > child { property1: value; property2: value; }
```

示例：

```
<html>
    <head>
        <style type="text/css">            #test>li{padding-left:30px;}
        </style>
    </head>
    <body>
        <ul id="test">
            <li>A</li>
            <li>B</li>
            <li>C
                <ul>
                    <li>C1</li>
                    <li>C2</li>
                    <li>C3</li>
                </ul>
            </li>
            <li>D</li>
        </ul>
    </body>
</html>
```

效果图：

- A
- B
- C
  - C1
  - C2
  - C3
- D

```
#test>li{padding-left:30px;}
```

这行代码选择了 id 为 test 的子元素 li，并将 padding-left 设置为30px，可以看到 li 标签内部的无序列表项目没有发生变化。若将以上代码改为

```
#test li{padding-left:30px;}
```

再来看一下效果图：

- A
- B
- C
  - C1
  - C2
  - C3
- D

li 中无序列表项的 padding 也发生了相应改变。

Note: 只有 IE7机器更高版本浏览器才支持子选择器。

相邻兄弟选择器

相邻兄弟选择器可以选择同一个父元素下某个元素之后的元素，并对其应用样式。

示例：

```
<html>
    <head>
        <style type="text/css">            h1+p{color:Red;}
        </style>
    </head>
    <body>
        <h1>小胖搞 IT</h1>
        <p>一个胖子从楼上掉下来...</p>
        <p>然后.............</p>
        <p>奇迹发生了！</p>
        <p>他弹了起来！</p>
    </body>
</html>
```

效果图：

小胖搞IT

一个胖子从楼上掉下来...

然后.....

奇迹发生了！

他弹了起来！

```
h1+p{color:Red;}
```

这行代码选择了 h1元素的下一个元素 p，并将其设置为红色。

Note: 只有 IE7机器更高版本浏览器才支持子选择器。

一个子选择器和相邻兄弟选择器结合使用的例子：

```
<html>
    <head>
        <style type="text/css">            #test > h1 + p {color:Red;}
        </style>
    </head>
    <body>
        <div id="test">
            <h1>小胖搞 IT</h1>
            <p>一个胖子从楼上掉下来...</p>
            <p>然后.....</p>
            <p>奇迹发生了！</p>
            <p>他弹了起来！</p>
            <div>
                <h1>瘦子搞 IT</h1>
                <p>一个瘦子从楼上掉下来...</p>
                <p>然后.....</p>
                <p>奇迹没有发生！</p>
                <p>他摔死了！</p>
            </div>
        </div>
    </body>
</html>
```

效果图：

小胖搞IT

一个胖子从楼上掉下来...

然后.....

奇迹发生了！

他弹了起来！

瘦子搞IT

一个瘦子从楼上掉下来...

然后.....

奇迹没有发生！

他摔死了！

```
#test > h1 + p {color:Red;}
```

这行代码选择了 id 为 test 的元素的 h1子元素，再找到它的下一个兄弟元素 p，并设置为红色， <h1>瘦子搞 IT</h1> 不是 id 为 test 的 div 的子元素，故没有变化。

属性选择器

属性选择器可以根据某个属性是否存在或根据属性的值来寻找元素，并对其使用样式。

语法：

选择器	描述
[attribute]	用于选取带有指定属性的元素。
[attribute=value]	用于选取带有指定属性和值的元素。
[attribute~=value]	用于选取属性值中包含指定词汇的元素。
[attribute|=value]	用于选取带有以指定值开头的属性值的元素，该值必须是整个单词。
[attribute^=value]	匹配属性值以指定值开头的每个元素。
[attribute$=value]	匹配属性值以指定值结尾的每个元素。
[attribute*=value]	匹配属性值中包含指定值的每个元素。

示例：

```
<html>
    <head>
        <style type="text/css">            a[title]{font-size:30px;}
a[title="Fatty"]{color:Red;}            a[title~="Fatty"]
{font-weight:bold;}            a[title|="FattyDoIT"]
{font-style:italic;}
a[title^="F"]{text-decoration:line-through; }
a[title$="IT"]::before {content:url(star.png);}
a[title*="Do"]::after {content:url(heart.png);}
        </style>
    </head>
    <body>
        <a href="http://www.cnblogs.com/fattydoit/">小胖搞 IT</a><br>
        <a href="http://www.cnblogs.com/fattydoit/" title="FattyDoIT">
小胖搞 IT</a><br>
        <a href="http://www.cnblogs.com/fattydoit/" title="Fatty">小胖搞
IT</a><br>
        <a href="http://www.cnblogs.com/fattydoit/" title="Fatty Do IT">
小胖搞 IT</a>
    </body>
</html>
```

效果图：


```
a[title]{font-size:30px;}
```

这行代码选择了所有具有 title 属性的 a 元素，并将字体大小设置为30px；

```
a[title="Fatty"]{color:Red;}
```

这行代码选择 title 值为 Fatty 的 a 元素，并将字体颜色设置为红色；

```
a[title~="Fatty"] {font-weight:bold;}
```

这行代码选择 title 所有属性值中包含 Fatty 的 a 元素，并将字体加粗；

```
a[title|="FattyDoIT"] {font-style:italic;}
```

这行代码选择 title 值以 FattyDoIT 开头且是一个单词的 a 元素，并将字体改为斜体；

```
a[title^="F"]{text-decoration:line-through; }
```

这行代码选择 title 属性值以 F 开头的所有 a 元素，并设置 text-decoration 为 line-through;

```
a[title$="IT"]::before {content:url(star.png);}
```

这行代码选择 title 属性值以 IT 结尾的所有 a 元素，并在之前放置一张图片；

```
a[title*="Do"]::after {content:url(heart.png);}
```

这行代码选择 title 属性值中包含 Do 的所有 a 元素，并在之后放置一张图片。

原文 http://www.cnblogs.com/fattydoit/p/3494424.html?utm_source=tuicool

GitHub 中最火的开源项目及编程语言

GitHub 目前已经成为全球最流行的开源项目托管平台，目前托管在 GitHub 上的项目数量已经达到了1000万，而达到这一里程碑只用了不到4年的时间，这足以见得开源的趋势以及 GitHub 的受欢迎程度。

2012年8月，GitHub 在每个项目主页面中加入了 Star 功能，允许用户通过标注 Star 的形式来标记自己感兴趣的项目。

最火的开源项目

本文就来看看目前 GitHub 中 Star 数最多的开源项目是什么。**下面是 Star 数排名前20的项目（Star 数随时都在变化，以下为2013年12月23日统计的数据）。**

排名	项目	所用语言	Star 数
1	twbs/bootstrap	JavaScript	62111
2	jquery/jquery	JavaScript	27082
3	joyent/node	JavaScript	26352
4	h5bp/html5-boilerplate	CSS	23355
5	mbostock/d3	JavaScript	20715
6	rails/rails	Ruby	20284
7	FortAwesome/Font-Awesome	CSS	19506
8	bartaz/impress.js	JavaScript	18637

9	angular/angular.js	JavaScript	17994
10	jashkenas/backbone	JavaScript	16502
11	Homebrew/homebrew	Ruby	15065
12	zurb/foundation	JavaScript	14944
13	blueimp/jQuery-File-Upload	JavaScript	14312
14	harvesthq/chosen	JavaScript	14232
15	mrdoob/three.js	JavaScript	13686
16	vhf/free-programming-books	Unknown	13658
17	adobe/brackets	JavaScript	13557
18	robbyrussell/oh-my-zsh	Shell	13337
19	jekyll/jekyll	Ruby	13283
20	github/gitignore	Unknown	13128

最火的编程语言

Star 排名前十的项目中，使用 JavaScript 编写的项目就占了7位，**下表显示了 Star 数前5000的项目所使用的编程语言情况。**其中1-10列表示 Star 数排名在1-10位项目中，有多少使用该语言编写。

编程语言	1~10	1~100	1~1000	1~5000	该语言排名第一的项目
JavaScript	7	54	385	1605	twbs/bootstrap (1)
CSS	2	8	41	174	h5bp/html5-boilerplate (4)
Ruby	1	9	153	786	rails/rails (6)
Python		5	64	420	django/django (44)
Unknown		5	30	138	vhf/free-programming-books (15)
C++		4	22	108	textmate/textmate (35)
PHP		3	38	248	symfony/symfony (58)
Shell		3	19	89	robbyrussell/oh-my-zsh (18)
Objective-C		2	89	495	AFNetworking/AFNetworking (30)
C		2	31	185	torvalds/linux (25)

Go		2	13	61	dotcloud/docker (45)
Java		1	32	255	nathanmarz/storm (56)
VimL		1	23	66	mathiasbynens/dotfiles (57)
CoffeeScript		1	22	80	jashkenas/coffee-script (43)
Scala			13	46	playframework/playframework (178)
C#			8	65	SignalR/SignalR (205)
Clojure			2	37	technomancy/leiningen (361)
Perl			2	26	sitaramc/gitolite (138)
ActionScript			2	10	mozilla/shumway (606)
Emacs Lisp			1	20	technomancy/emacs-starter-kit (477)
Erlang			1	15	erlang/otp (568)
Haskell			1	12	jgm/pandoc (740)
TypeScript			1	4	bitcoin/bitcoin (161)
Assembly			1	3	jmechner/Prince-of-Persia-Apple-II (269)
Elixir			1	2	elixir-lang/elixir (666)
Objective-J			1	2	cappuccino/cappuccino (667)
Rust			1	1	mozilla/rust (225)
Vala			1	1	p-e-w/finalterm (282)
Julia			1	1	JuliaLang/julia (356)
Visual Basic			1	1	bmatzelle/gow (800)
TeX				6	ieure/sicp (2441)

R				5	johnmyleswhite/ML_for_Hackers (2125)
Lua				4	leafo/moonscript (3351)
PowerShell				3	chocolatey/chocolatey (1580)
Prolog				3	onyxfish/csvkit (3498)
XSLT				2	wakaleo/game-of-life (1093)
Matlab				2	zk00006/OpenTLD (1292)
OCaml				2	MLstate/opalang (1380)
Dart				2	dart-lang/spark (1463)
Groovy				2	Netflix/asgard (1489)
Lasso				1	symfony/symfony-docs (2047)
LiveScript				1	gkz/LiveScript (2226)
Scheme				1	eholk/harlan (2648)
Common Lisp				1	google/lisp-koans (2889)
XML				1	kswedberg/jquery-tmbundle (2972)
Mirah				1	mirah/mirah (2985)
Arc				1	arclanguage/anarki (3389)
DOT				1	cplusplus/draft (3583)
Racket				1	plt/racket (3761)
F#				1	fsharp/fsharp (4518)
D				1	D-Programming-Language/phobos (4719)
Ragel in Ruby				1	jgarber/redcloth

Host					(4829)
Puppet				1	ansible/ansible-examples (4979)

更多数据：

- Star 数最多的前5000个项目：Top 5000 Repositories（csv 文件）
- Star 数统计脚本：adereth/counting-stars（Clojure 语言）

原文 http://www.iteye.com/news/28612?utm_source=tuicool

Oracle 优化 Java 字符串内部表示

为了不断改善 Java 性能，Oracle 已经宣布自 Java 1.7.0_06开始更改 String 类中的字符串内部表示。

此次更改删除了 String 底层实现中的两个非静态字段，这样做有助于防止内存泄漏。

原来的 String 实现基于四个非静态字段。第一个是 char[]值，它包含组成 String 的字符。第二个是 int offset，它保存值数组中第一个字符的索引。第三个是 int count，它保存用到的字符数。第四个是 int hash，它保存 String 哈希码的缓存值。

Oracle 报告称，当调用 String.substring（）创建 String 时，原来的实现会产生性能问题。许多其它 API，如 Pattern.split（），都会在内部调用 substring（）。当 String.substring（）被调用时，它会引用原来的包含组成 String 的字符的内部 char[]。

原先的实现之所以采用这种设计方式是为了节省内存，因为子串仍然会引用原来的字符数据。除此之外，String.substring（）的运行时间是一个常量（O（1）），而不像新的实现那样，有一个线性（O（n））运行时间。

不过，如果应用程序从原来的长字符串中抽取一个短字符串，并随后丢弃原来的字符串，那么在这种情况下可能会产生内存泄漏。在这个场景中，仍然存在一个活引用，指向原来字符串底层那个原来较大的 char[]值，这可能会占据许多不再使用的数据字节。

在早期版本中，Oracle 建议在短字符串上调用 new String（String）构造函数来避免这种情况。那个 API 只复制所需的底层 char[]的一部分，从而解除新的较短的字符串与原来的较长的父字符串的关联。

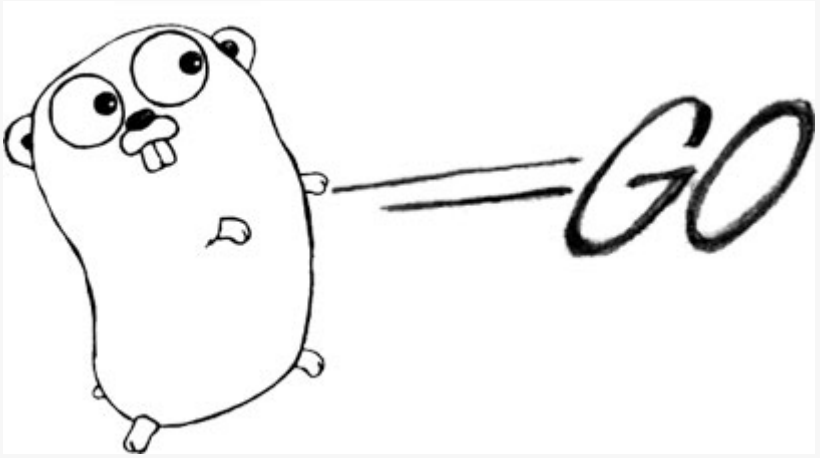
新规范删除了 String 的 offset 和 count 字段，因此子串不再共享底层的 char[] 值。

原文 http://www.infoq.com/cn/news/2013/12/Oracle-Tunes-Java-String?utm_source=tuicool

Google 抛弃 C 语言，采用 Go 语言重写 Go 编译器

Go 语言是 Google 开发的新型编程语言，将动态语言易于编写的特性和静态语言的高效性相结合，具备良好的易用性和极佳的执行效率。目前该语言已经发布了1.2正式版本。

Go 语言开发者 Russ Cox 近日透露，**在 Go 1.3版本之后，该语言的编译器将会使用 Go 语言重写。**



目前的 Go 编译器

目前 Go 编译器 gc 基于 C 语言，是从 Plan 9的 C 编译器衍生出来的，其中保持了原有的汇编程序、C 编译器和链接器，而部分针对 Go 的编译器（gc/6g/8g/5g）使用 C 语言进行了重写。

为何要采用 Go 语言实现

Russ Cox 指出，采用 Go 语言来实现编译器相比 C 语言的优势在于：

- 编写正确的 Go 代码要比编写正确的 C 代码容易
- 调试不正确的 Go 代码要比调试不正确的 C 代码容易

- Go 编译器必然需要对 Go 有个很好的了解，而使用 C 语言实现则增加了一个不必要的二次需求
- 与 C 相比，Go 语言对并行程序执行得更好
- 对于模块化、自动重写、单元测试以及分析，Go 比 C 有更好的标准支持
- 使用 Go 比使用 C 更加有趣

计划

Russ Cox 表示，将编译器从 C 实现转变为 Go 实现的过程，主要是通过自动翻译程序来进行。这一过程将分阶段进行：

- **第1阶段**：开发和调试翻译程序。
- **第2阶段**：使用翻译程序将编译器从 C 转换成 Go，并删除部分 C 语言副本。该阶段可能会在 Go 1.3版本中实现，可能仍会需要一些 C 代码。
- **第3阶段**：使用一些工具将编译器分割成包，并清理部分代码、添加文档、添加集成测试等。这个阶段将在 Go 1.4中实现，会将编译器彻底转换成为一个 Go 程序。
- **第4阶段**：通过标准的分析和测量技术对编译器的内存和 CPU 占用率进行优化，可能会引入并行处理。该阶段将在 Go 1.4中实现，部分特性可能会在 Go 1.5 中实现。
- **第5阶段**：使用最新版本的 Go 解析器和类型替换编译器前端。

舍弃的方案

对于为何不从头编写一个新的编译器，Russ Cox 表示，从头编写是一个愚蠢的行为，这意味着要抛弃之前很多人很多年以来的工作成果。

Go 语言开发团队还尝试了手动将 C 代码翻译为 Go 代码，但是在翻译了几个小型的 C/C++程序之后，他们发现这种方式比较繁琐，且容易出错，而且错误都非常隐蔽，不易发现。因此，他们决定首先开发出一个自动编译器，通过这种方式，出现的错误将会是一致的，容易查找。

原文 http://www.iteye.com/news/28602-Google-Go-Compiler

你需要知道的、有用的 Python 功能和特点

在使用 Python 多年以后，我偶然发现了一些我们过去不知道的功能和特性。

一些可以说是非常有用，但却没有充分利用。考虑到这一点，我编辑了一些的你应该了解的 Python 功能特色。

带任意数量参数的函数

你可能已经知道了 Python 允许你定义可选参数。但还有一个方法，可以定义函数任意数量的参数。

首先，看下面是一个只定义可选参数的例子

```
def function(arg1="",arg2=""):
    print "arg1: {0}".format(arg1)
    print "arg2: {0}".format(arg2)

function("Hello", "World")
# prints args1: Hello
# prints args2: World

function()
# prints args1:
# prints args2:
```

现在，让我们看看怎么定义一个可以接受任意参数的函数。我们利用元组来实现。

```
def foo(*args): # just use "*" to collect all remaining arguments into a tuple
    numargs = len(args)
    print "Number of arguments: {0}".format(numargs)
    for i, x in enumerate(args):
        print "Argument {0} is: {1}".format(i,x)

foo()
# Number of arguments: 0

foo("hello")
# Number of arguments: 1
# Argument 0 is: hello

foo("hello","World","Again")
# Number of arguments: 3
# Argument 0 is: hello
# Argument 1 is: World
# Argument 2 is: Again
```

使用 Glob()查找文件

大多 Python 函数有着长且具有描述性的名字。但是命名为 glob()的函数你可能不知道它是干什么的除非你从别处已经熟悉它了。
它像是一个更强大版本的 listdir()函数。它可以让你通过使用模式匹配来搜索文件。

```
import glob

# get all py files
files = glob.glob('*.py')
print files

# Output
# ['arg.py', 'g.py', 'shut.py', 'test.py']
```

你可以像下面这样查找多个文件类型：

```
import itertools as it, glob

def multiple_file_types(*patterns):
    return it.chain.from_iterable(glob.glob(pattern) for pattern in
patterns)

for filename in multiple_file_types("*.txt", "*.py"): # add as many
filetype arguements
    print filename

# output
#=========#
# test.txt
# arg.py
# g.py
# shut.py
# test.py
```

如果你想得到每个文件的绝对路径，你可以在返回值上调用 realpath()函数：

```
import itertools as it, glob, os

def multiple_file_types(*patterns):
    return it.chain.from_iterable(glob.glob(pattern) for pattern in
```

```
patterns)

for filename in multiple_file_types("*.txt", "*.py"): # add as many
filetype arguements
    realpath = os.path.realpath(filename)
    print realpath

# output
#=========#
# C:\xxx\pyfunc\test.txt
# C:\xxx\pyfunc\arg.py
# C:\xxx\pyfunc\g.py
# C:\xxx\pyfunc\shut.py
# C:\xxx\pyfunc\test.py
```

调试

下面的例子使用 inspect 模块。该模块用于调试目的时是非常有用的，它的功能远比这里描述的要多。

这篇文章不会覆盖这个模块的每个细节，但会展示给你一些用例。

```
import logging, inspect

logging.basicConfig(level=logging.INFO,

format='%(asctime)s %(levelname)-8s %(filename)s:%(lineno)-4d: %(messag
e)s',
    datefmt='%m-%d %H:%M',
    )
logging.debug('A debug message')
logging.info('Some information')
logging.warning('A shot across the bow')

def test():
    frame,filename,line_number,function_name,lines,index=\
        inspect.getouterframes(inspect.currentframe())[1]
    print(frame,filename,line_number,function_name,lines,index)

test()
```

```
# Should print the following (with current date/time of course)
#10-19 19:57 INFO     test.py:9    : Some information
#10-19 19:57 WARNING  test.py:10   : A shot across the bow
#(, 'C:/xxx/pyfunc/magic.py', 16, '', ['test()\n'], 0)
```

生成唯一 ID

在有些情况下你需要生成一个唯一的字符串。我看到很多人使用 md5()函数来达到此目的，但它确实不是以此为目的。

其实有一个名为 uuid()的 Python 函数是用于这个目的的。

```
import uuid
result = uuid.uuid1()
print result

# output => various attempts
# 9e177ec0-65b6-11e3-b2d0-e4d53dfcf61b
# be57b880-65b6-11e3-a04d-e4d53dfcf61b
# c3b2b90f-65b6-11e3-8c86-e4d53dfcf61b
```

你可能会注意到，即使字符串是唯一的，但它们后边的几个字符看起来很相似。这是因为生成的字符串与电脑的 MAC 地址是相联系的。

为了减少重复的情况，你可以使用这两个函数。

```
import hmac,hashlib
key='1'
data='a'
print hmac.new(key, data, hashlib.sha256).hexdigest()

m = hashlib.sha1()
m.update("The quick brown fox jumps over the lazy dog")
print m.hexdigest()

# c6e693d0b35805080632bc2469e1154a8d1072a86557778c27a01329630f8917
# 2fd4e1c67a2d28fced849ee1bb76e7391b93eb12
```

序列化

你曾经需要将一个复杂的变量存储在数据库或文本文件中吧？你不需要想一个奇特的方法将数组或对象格转化为式化字符串，因为 Python 已经提供了此功能。

```
import pickle
```

```
variable = ['hello', 42, [1,'two'],'apple']

# serialize content
file = open('serial.txt','w')
serialized_obj = pickle.dumps(variable)
file.write(serialized_obj)
file.close()

# unserialize to produce original content
target = open('serial.txt','r')
myObj = pickle.load(target)

print serialized_obj
print myObj

#output
# (lp0
# S'hello'
# p1
# aI42
# a(lp2
# I1
# aS'two'
# p3
# aaS'apple'
# p4
# a.
# ['hello', 42, [1, 'two'], 'apple']
```

这是一个原生的 Python 序列化方法。然而近几年来 JSON 变得流行起来，Python 添加了对它的支持。现在你可以使用 JSON 来编解码。

```
import json

variable = ['hello', 42, [1,'two'],'apple']
print "Original {0} - {1}".format(variable,type(variable))

# encoding
encode = json.dumps(variable)
print "Encoded {0} - {1}".format(encode,type(encode))
```

```
#deccoding
decoded = json.loads(encode)
print "Decoded {0} - {1}".format(decoded,type(decoded))

# output

# Original ['hello', 42, [1, 'two'], 'apple'] - <type 'list'="">
# Encoded ["hello", 42, [1, "two"], "apple"] - <type 'str'="">
# Decoded [u'hello', 42, [1, u'two'], u'apple'] - <type 'list'="">
```

这样更紧凑，而且最重要的是这样与 JavaScript 和许多其他语言兼容。然而对于复杂的对象，其中的一些信息可能丢失。

压缩字符

当谈起压缩时我们通常想到文件，比如 ZIP 结构。在 Python 中可以压缩长字符，不涉及任何档案文件。

```
import zlib

string =  """   Lorem ipsum dolor sit amet, consectetur
                adipiscing elit. Nunc ut elit id mi ultricies
                adipiscing. Nulla facilisi. Praesent pulvinar,
                sapien vel feugiat vestibulum, nulla dui pretium orci,
                non ultricies elit lacus quis ante. Lorem ipsum dolor
                sit amet, consectetur adipiscing elit. Aliquam
                pretium ullamcorper urna quis iaculis. Etiam ac massa
                sed turpis tempor luctus. Curabitur sed nibh eu elit
                mollis congue. Praesent ipsum diam, consectetur vitae
                ornare a, aliquam a nunc. In id magna pellentesque
                tellus posuere adipiscing. Sed non mi metus, at lacinia
                augue. Sed magna nisi, ornare in mollis in, mollis
                sed nunc. Etiam at justo in leo congue mollis.
                Nullam in neque eget metus hendrerit scelerisque
                eu non enim. Ut malesuada lacus eu nulla bibendum
                id euismod urna sodales. """

print "Original Size: {0}".format(len(string))

compressed = zlib.compress(string)
print "Compressed Size: {0}".format(len(compressed))
```

```
decompressed = zlib.decompress(compressed)
print "Decompressed Size: {0}".format(len(decompressed))

# output

# Original Size: 1022
# Compressed Size: 423
# Decompressed Size: 1022
```

注册 Shutdown 函数

有可模块叫 atexit，它可以让你在脚本运行完后立马执行一些代码。

假如你想在脚本执行结束时测量一些基准数据，比如运行了多长时间：

```
import atexit
import time
import math

def microtime(get_as_float = False) :
    if get_as_float:
        return time.time()
    else:
        return '%f %d' % math.modf(time.time())
start_time = microtime(False)
atexit.register(start_time)

def shutdown():
    global start_time
    print "Execution took: {0} seconds".format(start_time)

atexit.register(shutdown)

# Execution took: 0.297000 1387135607 seconds
# Error in atexit._run_exitfuncs:
# Traceback (most recent call last):
#   File "C:\Python27\lib\atexit.py", line 24, in _run_exitfuncs
#     func(*targs, **kargs)
# TypeError: 'str' object is not callable
# Error in sys.exitfunc:
```

```
# Traceback (most recent call last):
#   File "C:\Python27\lib\atexit.py", line 24, in _run_exitfuncs
#     func(*targs, **kargs)
# TypeError: 'str' object is not callable
```

打眼看来很简单。只需要将代码添加到脚本的最底层，它将在脚本结束前运行。但如果脚本中有一个致命错误或者脚本被用户终止，它可能就不运行了。
当你使用 atexit.register()时，你的代码都将执行，不论脚本因为什么原因停止运行。

结论

你是否意识到那些不是广为人知 Python 特性很有用？请在评论处与我们分享。谢谢你的阅读！

在使用 Python 多年以后，我偶然发现了一些我们过去不知道的功能和特性。一些可以说是非常有用，但却没有充分利用。考虑到这一点，我编辑了一些的你应该了解的 Pyghon 功能特色。

带任意数量参数的函数

你可能已经知道了 Python 允许你定义可选参数。但还有一个方法，可以定义函数任意数量的参数。
首先，看下面是一个只定义可选参数的例子

```
def function(arg1="",arg2=""):
    print "arg1: {0}".format(arg1)
    print "arg2: {0}".format(arg2)

function("Hello", "World")
# prints args1: Hello
# prints args2: World

function()
# prints args1:
# prints args2:
```

现在，让我们看看怎么定义一个可以接受任意参数的函数。我们利用元组来实现。

```
def foo(*args): # just use "*" to collect all remaining arguments into a tuple
    numargs = len(args)
    print "Number of arguments: {0}".format(numargs)
    for i, x in enumerate(args):
        print "Argument {0} is: {1}".format(i,x)
```

```
foo()
# Number of arguments: 0

foo("hello")
# Number of arguments: 1
# Argument 0 is: hello

foo("hello","World","Again")
# Number of arguments: 3
# Argument 0 is: hello
# Argument 1 is: World
# Argument 2 is: Again
```

使用 Glob()查找文件

大多 Python 函数有着长且具有描述性的名字。但是命名为 glob()的函数你可能不知道它是干什么的除非你从别处已经熟悉它了。
它像是一个更强大版本的 listdir()函数。它可以让你通过使用模式匹配来搜索文件。

```
import glob

# get all py files
files = glob.glob('*.py')
print files

# Output
# ['arg.py', 'g.py', 'shut.py', 'test.py']
```

你可以像下面这样查找多个文件类型：

```
import itertools as it, glob

def multiple_file_types(*patterns):
    return it.chain.from_iterable(glob.glob(pattern) for pattern in patterns)

for filename in multiple_file_types("*.txt", "*.py"): # add as many filetype arguements
    print filename
```

```
# output
#=========#
# test.txt
# arg.py
# g.py
# shut.py
# test.py
```

如果你想得到每个文件的绝对路径，你可以在返回值上调用 realpath() 函数：

```
import itertools as it, glob, os

def multiple_file_types(*patterns):
    return it.chain.from_iterable(glob.glob(pattern) for pattern in patterns)

for filename in multiple_file_types("*.txt", "*.py"): # add as many filetype arguements
    realpath = os.path.realpath(filename)
    print realpath

# output
#=========#
# C:\xxx\pyfunc\test.txt
# C:\xxx\pyfunc\arg.py
# C:\xxx\pyfunc\g.py
# C:\xxx\pyfunc\shut.py
# C:\xxx\pyfunc\test.py
```

调试

下面的例子使用 inspect 模块。该模块用于调试目的时是非常有用的，它的功能远比这里描述的要多。

这篇文章不会覆盖这个模块的每个细节，但会展示给你一些用例。

```
import logging, inspect

logging.basicConfig(level=logging.INFO,

format='%(asctime)s %(levelname)-8s %(filename)s:%(lineno)-4d: %(message)s',
```

```
    datefmt='%m-%d %H:%M',
    )
logging.debug('A debug message')
logging.info('Some information')
logging.warning('A shot across the bow')

def test():
    frame,filename,line_number,function_name,lines,index=\
        inspect.getouterframes(inspect.currentframe())[1]
    print(frame,filename,line_number,function_name,lines,index)

test()

# Should print the following (with current date/time of course)
#10-19 19:57 INFO     test.py:9    : Some information
#10-19 19:57 WARNING  test.py:10   : A shot across the bow
#(, 'C:/xxx/pyfunc/magic.py', 16, '', ['test()\n'], 0)
```

生成唯一 ID

在有些情况下你需要生成一个唯一的字符串。我看到很多人使用 md5() 函数来达到此目的，但它确实不是以此为目的。

其实有一个名为 uuid() 的 Python 函数是用于这个目的的。

```
import uuid
result = uuid.uuid1()
print result

# output => various attempts
# 9e177ec0-65b6-11e3-b2d0-e4d53dfcf61b
# be57b880-65b6-11e3-a04d-e4d53dfcf61b
# c3b2b90f-65b6-11e3-8c86-e4d53dfcf61b
```

你可能会注意到，即使字符串是唯一的，但它们后边的几个字符看起来很相似。这是因为生成的字符串与电脑的 MAC 地址是相联系的。

为了减少重复的情况，你可以使用这两个函数。

```
import hmac,hashlib
key='1'
data='a'
print hmac.new(key, data, hashlib.sha256).hexdigest()
```

```
m = hashlib.sha1()
m.update("The quick brown fox jumps over the lazy dog")
print m.hexdigest()

# c6e693d0b35805080632bc2469e1154a8d1072a86557778c27a01329630f8917
# 2fd4e1c67a2d28fced849ee1bb76e7391b93eb12
```

序列化

你曾经需要将一个复杂的变量存储在数据库或文本文件中吧？你不需要想一个奇特的方法将数组或对象格转化为式化字符串，因为 Python 已经提供了此功能。

```
import pickle

variable = ['hello', 42, [1,'two'],'apple']

# serialize content
file = open('serial.txt','w')
serialized_obj = pickle.dumps(variable)
file.write(serialized_obj)
file.close()

# unserialize to produce original content
target = open('serial.txt','r')
myObj = pickle.load(target)

print serialized_obj
print myObj

#output
# (lp0
# S'hello'
# p1
# aI42
# a(lp2
# I1
# aS'two'
# p3
# aaS'apple'
# p4
```

```
# a.
# ['hello', 42, [1, 'two'], 'apple']
```

这是一个原生的 Python 序列化方法。然而近几年来 JSON 变得流行起来，Python 添加了对它的支持。现在你可以使用 JSON 来编解码。

```
import json

variable = ['hello', 42, [1,'two'],'apple']
print "Original {0} - {1}".format(variable,type(variable))

# encoding
encode = json.dumps(variable)
print "Encoded {0} - {1}".format(encode,type(encode))

#deccoding
decoded = json.loads(encode)
print "Decoded {0} - {1}".format(decoded,type(decoded))

# output

# Original ['hello', 42, [1, 'two'], 'apple'] - <type 'list'="">
# Encoded ["hello", 42, [1, "two"], "apple"] - <type 'str'="">
# Decoded [u'hello', 42, [1, u'two'], u'apple'] - <type 'list'="">
```

这样更紧凑，而且最重要的是这样与 JavaScript 和许多其他语言兼容。然而对于复杂的对象，其中的一些信息可能丢失。

压缩字符

当谈起压缩时我们通常想到文件，比如 ZIP 结构。在 Python 中可以压缩长字符，不涉及任何档案文件。

```
import zlib

string =  """   Lorem ipsum dolor sit amet, consectetur
                adipiscing elit. Nunc ut elit id mi ultricies
                adipiscing. Nulla facilisi. Praesent pulvinar,
                sapien vel feugiat vestibulum, nulla dui pretium orci,
                non ultricies elit lacus quis ante. Lorem ipsum dolor
                sit amet, consectetur adipiscing elit. Aliquam
                pretium ullamcorper urna quis iaculis. Etiam ac massa
```

```
                sed turpis tempor luctus. Curabitur sed nibh eu elit
                mollis congue. Praesent ipsum diam, consectetur vitae
                ornare a, aliquam a nunc. In id magna pellentesque
                tellus posuere adipiscing. Sed non mi metus, at lacinia
                augue. Sed magna nisi, ornare in mollis in, mollis
                sed nunc. Etiam at justo in leo congue mollis.
                Nullam in neque eget metus hendrerit scelerisque
                eu non enim. Ut malesuada lacus eu nulla bibendum
                id euismod urna sodales. """

print "Original Size: {0}".format(len(string))

compressed = zlib.compress(string)
print "Compressed Size: {0}".format(len(compressed))

decompressed = zlib.decompress(compressed)
print "Decompressed Size: {0}".format(len(decompressed))

# output

# Original Size: 1022
# Compressed Size: 423
# Decompressed Size: 1022
```

注册 Shutdown 函数

有可模块叫 atexit，它可以让你在脚本运行完后立马执行一些代码。

假如你想在脚本执行结束时测量一些基准数据，比如运行了多长时间：

```
import atexit
import time
import math

def microtime(get_as_float = False) :
    if get_as_float:
        return time.time()
    else:
        return '%f %d' % math.modf(time.time())
start_time = microtime(False)
atexit.register(start_time)
```

```
def shutdown():
    global start_time
    print "Execution took: {0} seconds".format(start_time)

atexit.register(shutdown)

# Execution took: 0.297000 1387135607 seconds
# Error in atexit._run_exitfuncs:
# Traceback (most recent call last):
#   File "C:\Python27\lib\atexit.py", line 24, in _run_exitfuncs
#     func(*targs, **kargs)
# TypeError: 'str' object is not callable
# Error in sys.exitfunc:
# Traceback (most recent call last):
#   File "C:\Python27\lib\atexit.py", line 24, in _run_exitfuncs
#     func(*targs, **kargs)
# TypeError: 'str' object is not callable
```

打眼看来很简单。只需要将代码添加到脚本的最底层，它将在脚本结束前运行。但如果脚本中有一个致命错误或者脚本被用户终止，它可能就不运行了。
当你使用 atexit.register()时，你的代码都将执行，不论脚本因为什么原因停止运行。

原文
http://www.linuxeden.com/html/news/20131226/146876.html?utm_source=tuicool

VB 的未来计划

Lucian Wischik 回应了 Mads Torgersen 的演讲，提出了 Visual Basic 语言方面一些可能的变化。这些只是计划，一切还没有定数。这些变化主要是为了减少样板代码，并且没有提供我们在 VB 10或11中看到的重大改变。

只读属性

只读自动属性与 VB 语法可谓天作之合。只需要在属性声明前面加上 ReadOnly 关键字即可。它能保证只生成 getter，不过仍然可以在构造函数中设置该值。

```
ReadOnly Property Name As String
```

注释

VB 中的注释现在不能很好地与隐式续行符配合。因此 Lucian 列出的的第一个 VB 特性就是允许在隐式续行符后面使用注释。

字符串

Visual Basic 中的字符串目前不支持多行。因此首要的任务是允许这一点。其行为类似 C#的逐字字符串，但不需要前缀。

更有趣的特性是字符串插值（String Interpolation）。使用$"前缀，将不需要显式调用 String.Format。它还消除了因为算错替代变量的数目和位置而导致的异常。例如下面的代码：

```
Dim query = $"http://{ url }?name={ Escape(name) }&id={ Escape(id) }&o=xml"
```

在差不多4年前，Miguel de Icaza 就曾提议在 C#中支持字符串插值，并构建了一个工作原型。

字面量

Visual Basic 的日期字面量基于美国通用标准，这对于非美国开发者是相当不公平的。因此新的计划中可以允许 ISO 格式化的日期字面量。

很多语言中已经具备的二进制字面量也被列入计划当中，用&B 前缀表示。这对标志位风格的枚举来说是一个福音。

分部接口和模块

这项提议允许在 VB 中加入分部接口和模块，其用法与分部类相似。与分部类相同，这也是为了用于代码生成器。

空传播

与 C#类似，VB 团队也考虑提供空传播操作符。这样在调用方法之前就不再需要空验证。目前他们有两种选择，第一种与 C#一样。注意除了?.外，还有?()。

```
Dim y As Integer = x?.y?(3)?.z
```

如果?.或?(操作符的左边有空值，y 将得到默认值 z。

函数参数

与 C#类似，VB 也希望 params 关键字支持 IEnumerable 类型的参数，而不是只支持数组。

同样跟 C#类似的是，他们也希望能够在 out 实参中声明本地变量。

```
If Integer.TryParse(s, Out x) Then
```

这行代码会创建 x 变量，就好像它是在代码上面的语句中声明的。

他们也在考虑内联地声明其他变量。

```
If ( Dim x = GetValue()) > 15 Then Console .WriteLine(x)
```

其他方面

目前 VB 使用 IsNot 进行引用比较，但不能用于类型比较（如 TypeOf 操作符）。这项建议填补了这一空白。

原文 http://www.infoq.com/cn/news/2013/12/VB-Futures?utm_source=tuicool

C#转 C++的一点分享

从 C#转 C++有段时间了,一直想分享点什么,但又不太好意思分享,毕竟自己做 C++的时间不太长,最近感觉自己已能胜任现有工作,想分享的想法又强了点,前几天看到这样一篇博客《**那些年·我们读过的专业书籍**》，里面列了很多大家认为很好的书，加上自己在自学 C++的工程中也看了不少书，感觉并不是所有的书都值得花时间去看的，毕竟很多人一年下来也看不了2,3本书，不同的技术能力的人，适合看的书都不太一样，在这么多大家都认为是经典的书中，选出几本真正适合自己的才是王道，经典一多了，有些比起来就不是那么经典了，当然大家都说经典，自然有可看之处，如果有多余的时间，多看些书自然是好的。

下面是我看过的技术书籍（不一定看完），还有本《程序员的自我修养》在老家，其他几本不怎么样就没有列出来。

买的第一本技术书籍是《数据结构与算法分析》

当时刚毕业不久，在学校时没有买技术书籍的概念，在学校时也没怎么逛过技术网站，毕业后逛得比较多，当时是做 C#，那时 Android 很火想学，内心又一直认为成为一名好的程序员，底层知识是要会的，当时的情况是：在做 C#，想做 Android，又想做 C++，没准备长做 C#，最后买了这本算法相关的书籍，决定学 C++，注意准备开始吐槽：是谁说程序=算法+XX，这个等式对于很多刚毕业的同学们根本不成立，算法对于大部分的程序员来说都是弱项好不好，很多菜逼根本就不会算法，就会增删改查，有木有，别一开始就来最难的，容易打击自信心，是谁说算法很重要的，哥才写了90多篇博客，就有19篇跟算法有关的，私底下也学习了很多算法相关的文章，哥是要转 C++好不好。吐槽告一段落，**算法还是很重要的**，有时间还是要好好研究一下。

《Effective C++》买的第一本完全关于 C++的书

刚开始都看不懂，反复看之后，就很懂了，到现在为止应该至少看了4遍，感觉现在给我大半天的时间，能大致在看一遍，曾经一度想把书中的50条法则写成几篇博客的，最后由于太懒只写了3篇

Effective C++面向对象与继承

Effective C++ 类与函数设计和申明

Effective C++构造函数析构函数 Assignment 运算符

开始没看懂的时候，没觉得这书怎么样，等一条一条的看懂之后，发现这书他妈的写得太好，我忍不住的要赞美他，作者用很短的几句话就把事情说得很清楚，作者的表达能力的确牛逼，再次发现作者表达能力特别牛逼的是看吴军的《数学之美》，他用几句话就把一个关于图的问题讲得很清楚，其他的一些所谓的经典作者写的内容看起来就不是那么好理解，当然这跟讲的内容、跟读者所掌握的的知识以及读者的理解能力有关。

如果你要学 C++，那么我推荐你看这本，虽然它没有教我们入门 C++，也没有教我们深度理解 C++，也没讲更底层的知识，但是它是第一本，将很多关于 C++编程的重要知识点收录成的第一本书，第一本自然收录的是在开发中出现频率较大的，值得一讲的内容，之后有很多所谓经典的跟风之作，如

《Exceptional C++》和《More Exceptional C++》，包括作者自己的《More Effective C++》，

当然这些书中讲的内容可能是你八辈子也用不上的，但可能是要掌握的，这些书讲的差不多是一些技巧性的东西，或是实现了一个 XX 功能，有点通用性，人家把这些点讲得很清楚，这类书不用急着看，等你 C++学得差不多了，再看也不迟，那时会看得很快，因为他们没讲什么新的知识，都是对现有知识的运用。

《深度探索 C++对象模型》要想更深入的理解 C++，当然是这本，这本真的是经典中的必看。

这本书共7章320页，但出版社却把它弄得比较厚，好像不厚就感觉不是好书似的。每章的内容都很不错，建议重点看第三章和第四章，第三章讲 C++对象的内存布局，第四章讲 C++的各种方法编译后是啥样子的，以及方法是怎么被调用的，这是 C++最重要的东西，其他任何技术不都是建立在数据和方法之上吗！当然如果你是菜鸟

看完之后你还是菜鸟，你不会变成大牛，因为我看了这些书后我并没有变成大牛，你同样也不会，但你会更深入的理解 C++，你会对你自己的能力更自信。看完这本书后，我觉得我可以做 C++了，于是跨部门面试，十分钟左右，我现在的经理就问我什么时候可以过来。当然刚开始做 C++时各种不顺，实战太少。你不会因为看了几本书而成为大牛。看这本书之前除了《编译原理》还没有买，其他的书都看完或是看过一些，刚开始的时候有点看不懂，之后很快的看懂并看完了，因为看这本书之前我反复看了《深入理解计算机系统》的第七章（链接）和《程序员的自我修养》的大部分内容，对编译和链接还是有些了解，再看这本书自然会快些。看完这本书我写了三篇博客，但看的人很少。

构造函数产生的点及原因

虚方法的调用是怎么实现的(单继承 VS 多继承)

C++ Data Member 内存布局

《深入理解计算机系统》就不多说了，估计每个过来人都会推荐，如果你想成为一名好的程序员，就是必看必看，不管你是什么程序员，我相信很多 C#牛逼的程序员都看过，这本书讲了很多重要的知识，不是很深，但装逼足以，要全都看懂还是有些难的，之前就有个同事买了这本书，里面夹了一张跟他有纯洁男女关系的女性朋友的照片，作为书签,，这的确是督促自己看书的好方法。关于这本书我写了两篇博客，用这本书的内容写博客可以写很多篇。

数据对齐

在开发中你可能没有考虑到的两个性能优化

由于我一直在自学 C++，重点推荐这3本，前两本推荐指数更高：

1：深度探索 C++对象模型

2：深入理解计算机系统

3：Effective C++

其他的一些书都是值得看的，写到这里就有点不想写了，再说一本吧，其他的就不多说了，看完了上面这三本，再看其他的。由于我是做 Windows C++开发，就说说《Windows 核心编程》，做 Windows 开发的同学要看看。关于这本书我写了4篇博客

多线程编程--5种方法实现线程同步

Windows 几种线程同步方法介绍

Windows 线程基础

Windows 内核对象简介

这4篇博客基本上是讲线程、进程、内核对象、线程同步。这本书讲了 Windows 操作系统的很多东西，如内存管理，动态链接库，这些都是做 Windows 开发需要知道的，也是我比较薄弱的，在开发中基本都是用库，现在做的项目，两个 Solution，其中一个有100多个 project，刚开始来的时候经常编译都通过不了，加上每次编译的时间较长，搞了半天编译失败，挺打击人的。

关于书的内容就说到这里，免得大家以为我是来说书，显然说书不是我的强项，现在就来说说转 C++的一些感受。

老实说如果现在用的是 C#、JAVA、PHP 等，且没用过 C/C++不太建议转 C/C++，但

非常建议多看看底层的知识，C#做了几年之后发现也就是那点东西，不深入学习，很多东西其实是只会用，根本不知道到底是怎么回事。我相信很多 C#程序员不知道多态到底是怎么实现；方法到底是怎么被调用的；不知道哪些是在编译器确定，哪些是在运行时确定的；经常听到字符串池的概念等，微软把 C#封装得太好了，掉坑里的机会太少了，大不了不管3721，try...catch 一下，也让有些人变得不那么爱思考了，像 C/C++一不小心就掉坑里了，不思考都不行。经常听到有人说：C#入门容易，精通难。那是因为学底层的知识更难些好不好。说这些不是针对谁，以前写 C#就是吃着火锅唱着歌，各种轻松舒服，C++就两字：苦逼。要啥啥没有，如分割字符串这种最基本的功能都得自己实现，刚做 C++的感觉就是：自己坐在豪华的游艇上，看到对面的海盗船很酷，于是就往海盗船一跳，一上船发现这坐船到处都在漏水，各种设施都很差劲，一不小心就掉水里了，各种感觉不适应。

学 C++好很长时间吗？不要

从自学 C++到现在差不多3年时间，期间多次想放弃，心中常有的一个念头就是：把这些时间用在学 C#，C#可以学得更好，工资可以拿得更多，每次看到发到手中的这点工资，就没啥干劲了，但我也一直明白若要把程序当做一个长久的职业，我是要必会 C/C++的，期间偶尔有几个月没有学 C++，但最后还是放不下，又想起她。最后拖到两年多才做自己一直想做的 C++，感觉是个很漫长的过程，内心的针扎还是挺多的。这两年来业余时间拿来学习 C++的时间平均在半个小时左右，跟10000个小时比起来还是很少的，当然10000个小时是要成为真正的大牛的，而我是刚上贼船，看了这么多书理论还是有些，还需大量实战。

C++很难吗？NO

C++被成为世界上最难的编程语言，其实并没有传说中的那么难，但有一点是可以肯定的 C++程序员在没有成为小牛之前一直都在针扎，对，就是针扎，过得很苦逼，我在上个部门做 C#的时候没加过一天班，做 C++后，以前的同事说我瘦了，以前做 C#用一个月的时间一个人重写了公司的工作流，做 C++后，花了两个多月做个历史管理器，还经常加班。虽然现在自信多了，但还要继续针扎下去。等成为小牛后，我相信就不分编程语言了，但如果你不是做 C/C++，却想做 C/C++，就别听语言都是浮云这种废话。在你成为小牛之前你才是浮云，你得苦逼，而且这是转行，这也是我不太建议转 C/C++的一个原因，你之前学到的很多东西都是白学的，你要从0.1开始，又开始做菜鸟，大家都知道菜鸟的日子不是那么好过，等你成为小牛之后编程语言才是浮云。为什么大家都说他难呢？我估计就是被哪些没学好的人宣传的，没学好之前过得苦逼，那当然发自内心的认为难，其他的朋友们听他们一说估计也觉得难。在加上一些大牛也说 C++难，人家说难不是因为自己没学好而说难，人家是因为知道得大多，发现 C++太灵活，坑太多，新手很容易掉坑里。要我说，任何编程语言，基础语法也就那点东西，等你明白了这些基础的东西是怎么实现，为什么要这样实现，编译后是个什么样子，你也就发现 C++也就那样，不管牛不牛，也都是在用 C++干活，只不过是有的人再用 C++做增删改查，有的人在做大家都在用的产品。

怎么学习 C++？标准答案是：多思多写多看

作为新菜鸟的我，是不应该回答这个问题的，但这个问题的正确答案的确是多思多

写多看。现在来说说我的失败经历，作为智商不太低的我，学了两年多才做 C++，这显然是失败(哈哈，程序员就应该自信，千万别怀疑自己的智商，一点都不能有)。的确我是走了弯路，所以我像很多有失败经历的前辈一样，来分享自己的失败经历，以免后辈们重蹈前辈们的覆辙。

我学了好久的 C++，都不会写一个 C++的类，其实开始的时候一直都是在写 C，一直在用 C 写算法，严重受了算法重要的影响，其实开始的时候我都不是在学 C++，买的第一本技术书，不是关于 C++的，而是关于算法的。学 C++的时候也没有买一本比较好的系统介绍 C++的书籍。直接第一本 C++书籍就是 Effective C++，看这本书还是要一定基础的。看这本书的时候我已经做了两年多的 C#，在学校里有一点 C 的基础，之后也看过《高质量 C++编程指南》，还是有一定的程序基础，如果你有一定的程序基础，想转 C++，还是建议先看本系统介绍 C++的书籍，如《C++ Primer》，这本书的作者也是《深度探索 C++对象模型》的作者，还有 C++他爹写的那本啥也值得一读，大家都是这么说，应该不会错，两本选一本，有一定基础之后再看 Effective C++，感觉不错之后再看《深度探索 C++对象模型》，这本差不多之后再看《深入理解计算机系统》，这本不是讲 C++的，讲的是底层的东西，你学 C++不就是想学底层的东西吗，C++语法就那点东西，成为牛人之前还有一堆的书等着你去看，你还要写一堆的代码。

我现在在学习 Windows 的一些东西，做 Windows C++方向吗！业余时间看看《编译原理》，比较难懂，看懂之后，我相信很多东西都会明白得更透彻。明年估计是我成为小牛的重要阶段，阿汉加油。

为了不让我的博客因偏差太大，而让想做 C++的朋友走弯路，在这里付上耗子叔的《如何学好 C++语言》 http://coolshell.cn/articles/4119.html

原文 http://www.cnblogs.com/hlxs/p/3492048.html?utm_source=tuicool

IOS 怎么用 UIScrollView 来滚动和缩放他的内容第一篇

UIScrollView 是在 IOS 最有用的控件之一。他是一个来展现超过一个屏幕的内容的很好的方式。下面有很多的技巧来使用他。

这篇文章就是关于 UIScrollView 的，深入浅出，看看我们接下来学习的内容：

1：怎么用 UIScrollView 来展一个比较大的图片

2：当 UIScrollView 缩放的时候怎么一直保持在中间

3：在 UIScrollView 里面怎么嵌入一个复杂的视图层次

4：UIScrollView 的特性怎么和 UIPageControl 一起来浏览多个页面的内容

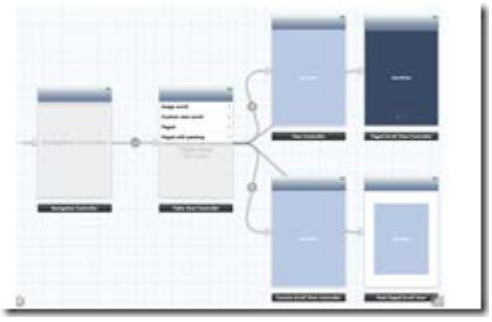
5：创建一个 UIScrollView 滚动视图在上面能看到下一页和上一页的一部分并且还能看到当前页面。这就像 appstore 的一个浏览 app 的一个效果。

这篇文章是 IOS5.0以上 xcode4.5的环境

我们开始创建一个项目如下图：

我们填上项目的名字还有你创建 appid 时候写的公司标识，还有类名字的前缀，设置我们的设备是 iPhone 我们暂时只支持 iPhone 的模式，选择单视图模版。选择下一步并且选择保存位置。

由于我们介绍 UIScrollView 的4个效果，因此我们创建一个 tableView，每个 cell 会出现一个新的视图控制器并且展现一个效果。



上面这个图显示现在你的 storyboard 是什么样的当你完成的时候。

我们编译 UITableView 的导航，接下来我们要做的是：

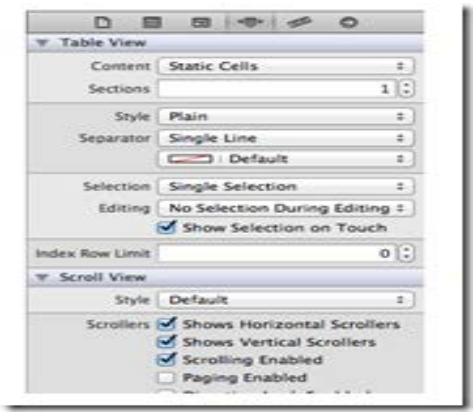
1：打开 MainStroyboard.storyboard 并且点击系统模版给我们创建的第一个初始化场景。

2：然后我们添加一个 UITableViewController 从对象库然后放到 storyboard。

3：现在选择 tableView 你刚才添加的然后选择 Editor，然后 Embed in，NavigationController。

4：选择 tableViewController 的 tableView，并且设置他的 cell 类型是静态类型的在属性检查器。

5：最后，设置 tableView 的 section 是一个，有4个 cell，设置 cell 是 basic 类型。然后把他们的 lables 改为 Image scroll，Custom View scroll，paged paged with peeking



保存这个 storyboard ，并且编译运行。你应该看到你的 tableView。如下图：

滚动缩放一个大图片：

我们接下来要做的是学习怎么用 UIScrollView 来缩放和滚动一个大的图片。

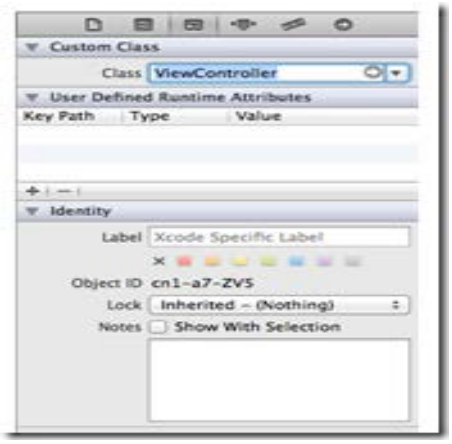
第一步你需要设置这个 Viewcontroller，选择 ViewController.h 并且添加一个 UIScrollView 的 outlet。让你的 Controller 符合 UIScrollView 的 UIScrollViewDelegate 协议如下

```
#import <UIKit/UIKit.h>


@interface ViewController : UIViewController <UIScrollViewDelegate>
@property (nonatomic, strong) IBOutlet UIScrollView *scrollView;


@end
```

然后在在 ViewController.m 设置实现属性

```
@synthesize scrollView = _scrollView;
```

回到 storyboard，从对象库拖拽一个 Viewcontroller 并且设置他的类是 Viewcontroller。



点击 tableview 的 Cell crl+点击鼠标左键向一个新的 Viewcontroller 拖拽，并且弹出一个 storyboard segues 并且选择 push 效果。

从对象库拖拽一个 UIScrollView 到 Viewcontroller 上并且填充。

然后然后把 UIScrollView 的输出口连上还有设置 Viewcontroller 作为 UIScrollView 的代理。如下图：

现在在 Viewcontroller.m 中的延展中添加属性和方法。这些属性方法是私有的。

然后添加

```
@synthesize imageView = _imageView;
```

现在我们开始设置我们的 UIScrollView 了在 viewDidLoad 和 viewVillAppear

用下面代码：

```
- (void)viewDidLoad {
    [super viewDidLoad];

    // 1
    UIImage *image = [UIImage imageNamed:@"photo1.png"];
    self.imageView = [[UIImageView alloc] initWithImage:image];
    self.imageView.frame = (CGRect){.origin=CGPointMake(0.0f,
0.0f), .size=image.size};
    [self.scrollView addSubview:self.imageView];
```

```
    // 2
    self.scrollView.contentSize = image.size;

    // 3
    UITapGestureRecognizer *doubleTapRecognizer =
[[UITapGestureRecognizer alloc] initWithTarget:self
action:@selector(scrollViewDoubleTapped:)];
    doubleTapRecognizer.numberOfTapsRequired = 2;
    doubleTapRecognizer.numberOfTouchesRequired = 1;
    [self.scrollView addGestureRecognizer:doubleTapRecognizer];

    UITapGestureRecognizer *twoFingerTapRecognizer =
[[UITapGestureRecognizer alloc] initWithTarget:self
action:@selector(scrollViewTwoFingerTapped:)];
    twoFingerTapRecognizer.numberOfTapsRequired = 1;
    twoFingerTapRecognizer.numberOfTouchesRequired = 2;
    [self.scrollView addGestureRecognizer:twoFingerTapRecognizer];
}

- (void)viewWillAppear:(BOOL)animated {
    [super viewWillAppear:animated];

    // 4
    CGRect scrollViewFrame = self.scrollView.frame;
    CGFloat scaleWidth = scrollViewFrame.size.width /
self.scrollView.contentSize.width;
    CGFloat scaleHeight = scrollViewFrame.size.height /
self.scrollView.contentSize.height;
    CGFloat minScale = MIN(scaleWidth, scaleHeight);
    self.scrollView.minimumZoomScale = minScale;

    // 5
    self.scrollView.maximumZoomScale = 1.0f;
    self.scrollView.zoomScale = minScale;

    // 6
    [self centerScrollViewContents];
}
```

上面的代码看起来有点复杂。因此我们停下来一步步的分析下。

1：第一步，你需要创建一个 UIImageView，设置他的 Image 属性，然后设置 UIImageView 的坐标，并且添加到 UIScrollView 上

2：然后我们设置 UIScrollView 的 contentSize，这样做的目的是让 UIScrollView

知道他自己能向横方向和竖方向滚动多远或者说多少像素。

3：然后向 UIScrollView 上面添加了两个手势:一个是双击手势来缩小，另一个两个手指单击来放大。

4：接下来我们需要计算 UIScrollView 的最小缩放比例。缩放比例是1意味着 UIScrollView 的内容是正常大小展示。小于1，展示内容放大，当

大于1说明内容缩小。为了得到最小缩放比例，你需要计算你缩放多少才能让图片舒适的展示到 UIScrollView 里根据他的宽度。然后你根据他的高度做相同的计算。最后比较这两个缩放比例的最小的一个设置为 UIScrollView 最小缩放比例。给你一个缩放比例然后你可以看到整张图片当放大的时候。

5:你设置最大缩放比例为1，因为缩放的比图片分辨率大你看图片会模糊。你设置初始缩放为最小缩放比例。因此这个图片可以开始充分放大。

6：让你的图片永远在 UIScrollView 中间当缩放时候。

```
- (void)centerScrollViewContents {
    CGSize boundsSize = self.scrollView.bounds.size;
    CGRect contentsFrame = self.imageView.frame;

    if (contentsFrame.size.width < boundsSize.width) {
        contentsFrame.origin.x = (boundsSize.width -
contentsFrame.size.width) / 2.0f;
    } else {
        contentsFrame.origin.x = 0.0f;
    }

    if (contentsFrame.size.height < boundsSize.height) {
        contentsFrame.origin.y = (boundsSize.height -
contentsFrame.size.height) / 2.0f;
    } else {
        contentsFrame.origin.y = 0.0f;
    }

    self.imageView.frame = contentsFrame;
}
```

如果 UIScrollView 的 bounds 大小大于 UIImageView 图片 frame 的大小，那么图片的坐标就是条件为真时计算的结果，相反就是原始坐标。

```
- (void)scrollViewDoubleTapped:(UITapGestureRecognizer*)recognizer {
    // 1
    CGPoint pointInView = [recognizer locationInView:self.imageView];

    // 2
    CGFloat newZoomScale = self.scrollView.zoomScale * 1.5f;
```

```
    newZoomScale = MIN(newZoomScale, self.scrollView.maximumZoomScale);

    // 3
    CGSize scrollViewSize = self.scrollView.bounds.size;

    CGFloat w = scrollViewSize.width / newZoomScale;
    CGFloat h = scrollViewSize.height / newZoomScale;
    CGFloat x = pointInView.x - (w / 2.0f);
    CGFloat y = pointInView.y - (h / 2.0f);

    CGRect rectToZoomTo = CGRectMake(x, y, w, h);

    // 4
    [self.scrollView zoomToRect:rectToZoomTo animated:YES];
}
```

1：获得你点击图片的坐标位置.
2:接下来计算缩放比例缩放150%，但是必须限制最大缩放比例
3：然后用第一步计算的位置计算你想要缩放的位置大小。
4：最后，你需要告诉 UIScrollView 缩放的 frame 并且加上动画。

```
- (void)scrollViewTwoFingerTapped:(UITapGestureRecognizer*)recognizer {
    // Zoom out slightly, capping at the minimum zoom scale specified by the scroll view
    CGFloat newZoomScale = self.scrollView.zoomScale / 1.5f;
    newZoomScale = MAX(newZoomScale, self.scrollView.minimumZoomScale);
    [self.scrollView setZoomScale:newZoomScale animated:YES];
}
```

这类似放大的方式。

```
- (UIView*)viewForZoomingInScrollView:(UIScrollView *)scrollView {
    // Return the view that you want to zoom
    return self.imageView;
}
```

这是 UIScrollView 缩放机制的灵魂地方。当 UIScrollView 完成缩放时候你告诉他是哪个视图在 UIScrollView 里面实现了缩放。

```
- (void)scrollViewDidZoom:(UIScrollView *)scrollView {
    // The scroll view has zoomed, so you need to re-center the contents
    [self centerScrollViewContents];
}
```

这个方法是当UIScrollView完成缩放时候,你需要通知视图在UIScrollView中间,否则 UIScrollView 缩放不自然。

编译运行项目出现上图效果，你可以试着放大缩小滚动。

原文 http://www.cnblogs.com/qiqibo/p/3496223.html?utm_source=tuicool

Beanstalkd 队列 概述

本文对 beanstalkd 的介绍简短精炼，适合初次接触消息队列并想使用 beanstalk 的人。在我看过的一些介绍资料里，这篇文章介绍地最为精炼，本文大多内容修改自它:P。我也是第一次接触消息队列，使用 beanstalkd 的原因是组里比较资深的经理推荐，说道 beanstalkd 在某些大公司内都会使用，很轻量级，性能很高(是纯 C 写的)。

基本特性

同 RabbitMQ，ZeroMQ 相比，Beanstalkd 是一个更加轻量级的消息队列，也可以理解为是一个工作队列，本质上是基于多个 tube 的 workqueue，queue 里存放的是 job，每个 job 带一个递增的唯一 id，以及消息体(byte[])。

Beanstalkd 的 job 状态多样化，支持任务优先级 (priority)，延时 (delay)，超时重发 (time-to-run) 和预留 (buried)，能够很好的支持分布式的后台任务和定时任务处理。

它的内部实现采用 libevent，服务器-客户端之间用类似 memcached 的轻量级通讯协议，因此有很高的性能。

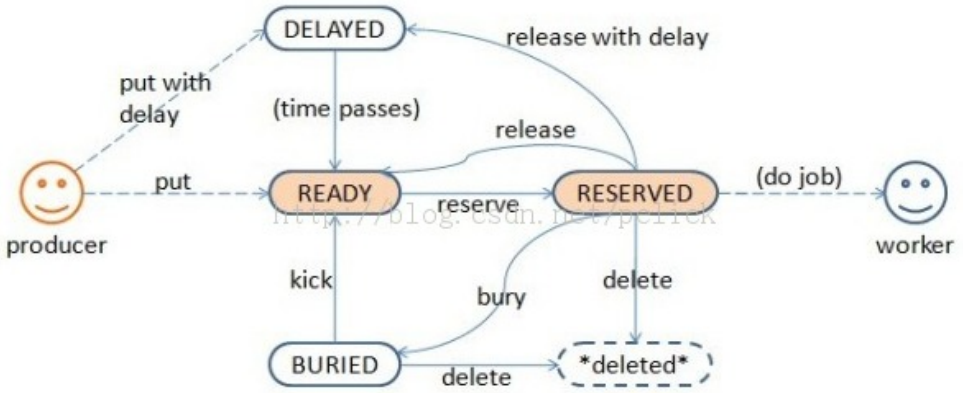
beanstalkd 提供了 binlog 机制，启动的时候可以打开此功能。当重启 beanstalkd 时，当前任务状态能够从纪录的本地 binlog 中恢复。虽然没有单点的 HA，但是带了容错。

两个重要概念

tube 类似于消息主题（topic），在一个 Beanstalkd 中可以支持多个管道， 每个管道都有自己的发布者（producer）和消费者（consumer），管道之间可以做到互不影响。

job 真正的消息体，带有唯一 id(从1开始递增)和一个 body(byte 流)，有丰富的可控状态，设计地非常好用。

下图详细阐述了 job 状态的流转流程，理解清楚下图对于使用 beanstalkd 有重要意义。



READY - 需要立即处理的任务，当延时（DELAYED）任务到期后会自动成为当前任务
DELAYED - 延迟执行的任务，当消费者处理任务后，可以用将消息再次放回 DELAYED 队列延迟执行
RESERVED - 已经被消费者获取，正在执行的任务(将不会被别的消费者拿到)。Beanstalkd 负责检查任务是否在 TTR(time-to-run) 内完成；
BURIED - 保留的任务：任务不会被执行，也不会消失，除非有人把它 "踢" 回队列；
DELETED - 消息被彻底删除。Beanstalkd 不再维持这些消息。

以上这些状态可以通过实际启动 beanstalkd 后，用 telnet 连接并执行一些 put，reserve 操作或者直接编程体验的方式得到更好的理解。

其他概念

任务优先级（priority）
任务（job）可以有 0~2^32 个优先级，0 代表最高优先级。 beanstalkd 采用最大最小堆（Min-max heap）处理任务优先级排序，任何时刻调用 reserve 命令的

消费者总是能拿到当前优先级最高的任务，时间复杂度为 O(logn).

延时任务（delay）

有两种方式可以延时执行任务（job）：生产者发布任务时指定延时；或者当任务处理完毕后，消费者再次将任务放入队列延时执行（RELEASE with <delay>）。这种机制可以实现分布式的 java.util.Timer，这种分布式定时任务的优势是：如果某个消费者节点故障，任务超时重发（time-to-run）能够保证任务转移到另外的节点执行。

任务超时重发（time-to-run）

Beanstalkd 把任务返回给消费者以后：消费者必须在预设的 TTR（time-to-run）时间内发送 delete / release/ bury 改变任务状态；否则 Beanstalkd 会认为消息处理失败，然后把任务交给另外的消费者节点执行。如果消费者预计在 TTR（time-to-run）时间内无法完成任务，也可以发送 touch 命令，它的作用是让 Beanstalkd 从系统时间重新计算 TTR（time-to-run).

任务预留（buried）

如果任务因为某些原因无法执行，消费者可以把任务置为 buried 状态让 Beanstalkd 保留这些任务。管理员可以通过 peek buried 命令查询被保留的任务，并且进行人工干预。简单的，kick <n> 能够一次性把 n 条被保留的任务踢回队列。

Beanstalkd 协议

Beanstalkd 采用类 memcached 协议，客户端通过文本命令与服务器交互。这些命令可以简单的分成三组:

生产类 - use <tube> / put <priority> <delay> <ttr> [bytes]:

生产者用 use 选择一个管道（tube），然后用 put 命令向管道发布任务（job).

消费类 - watch <tubes> / reserve / delete <id> / release <id> <priority> <delay> / bury <id> / touch <id>

消费者用 watch 选择多个管道（tube），然后用 reserve 命令获取待执行的任务，这个命令是阻塞的。客户端直到有任务可执行才返回。当任务处理完毕后，消费者可以彻底删除任务（DELETE），释放任务让别人处理（RELEASE），或者保留（BURY）任务。

维护类 - peek job / peek delayed / peek ready / peek buried / kick <n>

用于维护管道内的任务状态，在不改变任务状态的条件下获取任务。可以用消费类

命令改变这些任务的状态。
被保留（buried）的任务可以用 kick 命令 "踢" 回队列。

java 客户端

在官方推荐的几个 java 的客户端里，我选择了 TrendrrBeanstalk，是基于 MIT 许可证的开源项目，源码一共六个.java 文件。用起来很简单，但是没有封装 peek 相关的操作。如果你有更好的 java client 的话欢迎交流沟通哈 :) 随意来个 example。

[java] view plaincopy

```
public class BeanstalkExample {

    protected static Log log = LogFactory.getLog(BeanstalkExample.class);

    public static void main(String[] args) {
        try {
            clientExample();
            //pooledExample();
        } catch (BeanstalkException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }
    }

    /**
     * Example for using an unpooled client
     *
     * @throws BeanstalkException
     */
    public static void clientExample() throws BeanstalkException {
        BeanstalkClient client = new BeanstalkClient("ip", 8300, "example");

        //client.put(1l, 0, 5000, "this is some data".getBytes());
        log.info(client.tubeStats());
        //while(client.reserve(60) != null) {
            BeanstalkJob job = client.reserve(60);
            log.info("Get job: " + job.getId());
            client.deleteJob(job);
        //}

        client.close(); // closes the connection
```

```
    }

    public static void pooledExample() throws BeanstalkException {
        BeanstalkPool pool = new
BeanstalkPool("jx-crm-flare00.jx.baidu.com", 8300, 30, // poolsize
                "example" // tube to use
        );

        BeanstalkClient client = pool.getClient();

        client.put(1l, 0, 5000, "this is some data".getBytes());
        BeanstalkJob job = client.reserve(60);

        client.deleteJob(job);

        client.close(); // returns the connection to the pool
    }
}
```

原文
http://blog.csdn.net/pelick/article/details/17590557?utm_source=tuicool

Android 猜牌小游戏（改进版）

此篇基于 Android 猜牌小游戏 修改，简化代码，逻辑更为合理。

只修改 MainActivity.java 部分的代码：

Java 代码

```
package com.ming.puke;

import java.util.ArrayList;
import java.util.Random;
import android.app.Activity;
import android.os.Bundle;
import android.view.View;
import android.view.View.OnClickListener;
import android.widget.Button;
import android.widget.ImageView;
import android.widget.Toast;

public class MainActivity extends Activity {
    private Button btnplayagian;// 再玩一次
    private boolean isopened = false;// 判断是否已开牌
```

```
22      private int[] pks = { R.drawable.p01, R.drawable.p02,
R.drawable.p03 };// 存放3张扑克的值
23      private ArrayList<ImageView> pkimages = new
ArrayList<ImageView>();// 存放3张扑克的体
24
25      protected void onCreate(Bundle savedInstanceState) {
26          super.onCreate(savedInstanceState);
27          setContentView(R.layout.main);
28          pkimages.add((ImageView) findViewById(R.id.puke1));
29          pkimages.add((ImageView) findViewById(R.id.puke2));
30          pkimages.add((ImageView) findViewById(R.id.puke3));
31          for (int i = 0; i < pkimages.size(); i++) {
32              pkimages.get(i).setOnClickListener(l);
33          }
34          btnplayagian = (Button) findViewById(R.id.btnplayagian);
35          btnplayagian.setOnClickListener(l);
36          randomPuke();
37      }
38
39      OnClickListener l = new OnClickListener() {
40          public void onClick(View v) {
41              if (v.getId() == R.id.btnplayagian) {
42                  randomPuke();
43              } else {
44                  if (!isopened) {// 如果牌还没开才能开牌
45                      for (int i = 0; i < pkimages.size(); i++) {
46                          pkimages.get(i).setImageResource(pks[i]);
47                          if (i != pkimages.indexOf(v)) {// 将玩家没
选择的牌面以灰暗效果处理
48                              pkimages.get(i).setAlpha(100);// 设置
透明度，取值范围为0~255，数值越小越透明。
49                          }
50                      }
51                      Toast.makeText(
52                              // 显示开牌结果
53                              MainActivity.this,
54                              R.drawable.p01 ==
pks[pkimages.indexOf(v)] ? "恭喜你猜对了"
55                                      : "遗憾你猜错了",
Toast.LENGTH_SHORT).show();
56                      isopened = true;
57                  }
58              }
59          }
60      };
61
62      /** 洗牌 */
63      public void randomPuke() {
```

```
64          for (int i = 0; i < pkimages.size(); i++) {// 随机替换牌的顺序（并将洗好的牌翻到背面）
65              pkimages.get(i).setImageResource(R.drawable.p04);
66              int item = new Random().nextInt(3);// 抽出第 item 张牌
67              pkimages.get(i).setAlpha(255);// 初始化选中状态
68              pks[item] = pks[0] + 0 * (pks[0] = pks[item]);// 将第 item 张牌与第1张牌替换
69          }
70          isopened = false;
71      }
72  }
```

原文 http://1119495352.iteye.com/blog/1996910?utm_source=tuicool

Android 开发心得——网页通过 webview 调用 Android 的图片或文件选择

前段时间因为客户需求，做一个客户端结合 web 微网站的应用。其中，这个应用设计到了要修改头像，但是这个页面却是在微网站上实现的，意味着网站要调用到 Android 的打开文件的方法，那么这个通过 webview 是怎么实现的呢？

经过跟服务器的同事讨论发现，方法都是跟 pc 上是一样的，都是调用一个叫 input type=file 的属性，于是我就开始找，webview 是怎么响应这个属性的了。

于是翻网站找到资料，不难查到，想要适用 php 上调用打开文件的方法，webview 就要重写一个名为 openFileChooser 的方法。

但是这个方法的使用却不简单,这个方法是要调用 webview 的 setWebChromeClient 方法，然后重写一个 WebChromeClient 类。来到这一步，相信有点开发经验的同行都不难解决。问题的关键就在于,当你重写 WebChromeClient 这个类的时候会发现，根本就没有 openFileChooser 这个方法,那要怎么重写呢？是不是意味着这个方法其实行不通？于是再次翻查资料，发现原来这个方法居然是隐藏方法，并不不存在显性的继承重写关系。

最后，我发现要使用这个方法，还得自己继承 WebChromeClient 这个类把 openFileChooser(ValueCallback<Uri> uploadFile)这个方法给写出来，代码如下：

```
abstract class TestWebChromeClientextends WebChromeClient
{

    private WebChromeClient mWrappedClient;

    protected TestWebChromeClient(WebChromeClient wrappedClient)
    {
        mWrappedClient = wrappedClient;
    }

    /** {@inheritDoc} */
    @Override
    public void onProgressChanged(WebView view,int newProgress)
    {
        mWrappedClient.onProgressChanged(view, newProgress);
    }

    /** {@inheritDoc} */
    @Override
    public void onReceivedTitle(WebView view, String title)
    {
        mWrappedClient.onReceivedTitle(view, title);
    }

    /** {@inheritDoc} */
    @Override
    public void onReceivedIcon(WebView view, Bitmap icon)
    {
        mWrappedClient.onReceivedIcon(view, icon);
    }

    /** {@inheritDoc} */
    @Override
    public void onReceivedTouchIconUrl(WebView view, String
url,boolean precomposed)
    {
        mWrappedClient.onReceivedTouchIconUrl(view, url,
precomposed);
    }

    /** {@inheritDoc} */
    @Override
    public void onShowCustomView(View view, CustomViewCallback
callback)
    {
        mWrappedClient.onShowCustomView(view, callback);
    }
```

```
48
49      /** {@inheritDoc} */
50      @Override
51      public void onHideCustomView()
52      {
53          mWrappedClient.onHideCustomView();
54      }
55
56      /** {@inheritDoc} */
57      @Override
58      public    boolean    onCreateWindow(WebView    view,boolean
59  dialog,boolean userGesture,
60              Message resultMsg)
61      {
62          return    mWrappedClient.onCreateWindow(view,    dialog,
63  userGesture, resultMsg);
64      }
65
66      /** {@inheritDoc} */
67      @Override
68      public void onRequestFocus(WebView view)
69      {
70          mWrappedClient.onRequestFocus(view);
71      }
72
73      /** {@inheritDoc} */
74      @Override
75      public void onCloseWindow(WebView window)
76      {
77          mWrappedClient.onCloseWindow(window);
78      }
79
80      /** {@inheritDoc} */
81      @Override
82      public boolean onJsAlert(WebView view, String url, String
83  message, JsResult result)
84      {
85          return  mWrappedClient.onJsAlert(view,  url,  message,
86  result);
87      }
88
89      /** {@inheritDoc} */
90      @Override
91      public boolean onJsConfirm(WebView view, String url, String
92  message, JsResult result)
93      {
94          return  mWrappedClient.onJsConfirm(view,  url,  message,
```

```
result);
    }

    /** {@inheritDoc} */
    @Override
    public boolean onJsPrompt(WebView view, String url, String
message,
            String defaultValue, JsPromptResult result)
    {
        return mWrappedClient.onJsPrompt(view, url, message,
defaultValue, result);
    }

    /** {@inheritDoc} */
    @Override
    public boolean onJsBeforeUnload(WebView view, String url,
String message,
            JsResult result)
    {
        return    mWrappedClient.onJsBeforeUnload(view,    url,
message, result);
    }

    /** {@inheritDoc} */
    @Override
    public void onExceededDatabaseQuota(String url, String
databaseIdentifier,
            long      currentQuota,long      estimatedSize,long
totalUsedQuota,
            WebStorage.QuotaUpdater quotaUpdater)
    {
        mWrappedClient.onExceededDatabaseQuota(url,
databaseIdentifier, currentQuota,
                estimatedSize, totalUsedQuota, quotaUpdater);
    }

    /** {@inheritDoc} */
    @Override
    public void onReachedMaxAppCacheSize(long spaceNeeded,long
totalUsedQuota,
            WebStorage.QuotaUpdater quotaUpdater)
    {
        mWrappedClient
                .onReachedMaxAppCacheSize(spaceNeeded,
totalUsedQuota, quotaUpdater);
    }

```

```
142     /** {@inheritDoc} */
143     @Override
144     public void onGeolocationPermissionsShowPrompt(String origin,
145             GeolocationPermissions.Callback callback)
146     {
147 
148 mWrappedClient.onGeolocationPermissionsShowPrompt(origin,
149 callback);
150     }
151 
152     /** {@inheritDoc} */
153     @Override
154     public void onGeolocationPermissionsHidePrompt()
155     {
156         mWrappedClient.onGeolocationPermissionsHidePrompt();
157     }
158 
159     /** {@inheritDoc} */
160     @Override
161     public boolean onJsTimeout()
162     {
163         return mWrappedClient.onJsTimeout();
164     }
165 
166     /** {@inheritDoc} */
167     @Override
168     @Deprecated
169     public void onConsoleMessage(String message, int lineNumber,
170 String sourceID)
171     {
172         mWrappedClient.onConsoleMessage(message, lineNumber,
173 sourceID);
174     }
175 
176     /** {@inheritDoc} */
177     @Override
178     public boolean onConsoleMessage(ConsoleMessage
179 consoleMessage)
180     {
181         return mWrappedClient.onConsoleMessage(consoleMessage);
182     }
183 
184     /** {@inheritDoc} */
185     @Override
186     public Bitmap getDefaultVideoPoster()
187     {
188         return mWrappedClient.getDefaultVideoPoster();
```

```
189     }
190 
        /** {@inheritDoc} */
        @Override
        public View getVideoLoadingProgressView()
        {
            return mWrappedClient.getVideoLoadingProgressView();
        }

        /** {@inheritDoc} */
        @Override
        public    void    getVisitedHistory(ValueCallback<String[]>
    callback)
        {
            mWrappedClient.getVisitedHistory(callback);
        }

        /** {@inheritDoc} */

        public void openFileChooser(ValueCallback<Uri> uploadFile)
        {
            ((TestWebChromeClient)
    mWrappedClient).openFileChooser(uploadFile);
        }

    }
```

以上代码我是在 eoe 上发现的，并不是我自己参详发现的哈。但是原作是谁我就不知道了，因为这段代码你只要百度一下 openFileChooser 这个方法你就能找到。

当你这样写好，然后就是去设置 WebChromeClient 的时候了。设置方法如下：

```
1   mContentView.web_main_web.setWebChromeClient(new TestWebChromeClient(
2                   new WebChromeClient())
3   {
4   public void openFileChooser(ValueCallback<Uri> uploadFile)
5   {
6       if (mUploadMessage !=null)
7       return;
8       mUploadMessage = uploadFile;
9 
1       //自己写你的调用图片方法，我这里是封装了调用图片的方法的
0       //关键点在于这个 mUploadMessage 参数，获取到图片以后传给这个参数回
1   去就可以了。
1       //具体用法百度一下就有。
1 
```

```
Select_Activity.start(mContext,mUploadMessage,indexUrlString,FILE_SELECTED);
    }

});
```

当你设置完后，这个时候你就该兴高采烈地去测试了。

于是，你有可能会兴高采烈地发现，不行!! 完全没有反应！

是不是开始怀疑这个方法是坑人的？是不是在努力骂撸主？来来来，别生气，让我告诉你真相。

我告诉你哦，这个方法其实一点都不吭人，真的可以啊，不过~这是在3.0以前的 sdk 上有效...但是现在的主流 Android 机基本都是4.0以上了，哪里还有3.0以前的系统？

于是你又开始了新的一轮骂娘，为什么我会知道？因为我那个时候也是这个反应！

那时候我努力地翻资料，把百度、eoe、CSDN、德问、博客园、安卓巴士、DEVDIV 都翻烂了，终于找到了原因，原来泥煤的3.0的要多加一个参数才能生效！

于是我傻乎乎的仿照人家重写的 openFileChooser 方法，给 TestWebChromeClient 这个类添加了一个 openFileChooser(ValueCallback<Uri> uploadFile, String acceptType)方法。（具体这个 acceptType 参数有什么用，我还不怎么清楚，有知道的大神麻烦告知一下哈）在 webview 的 setWebChromeClient 方法里也添加了一个对应调用方法。

于是新一轮测试又开始了。

终于，你又一次兴高采烈地骂娘了，泥煤的还是不行啊！（po 主：喂！别打头，把我打傻了以后就不能分享技术了!）

于是，我终于相信了国内搜索引擎和论坛的不靠谱，我投靠了谷歌和 stackoverflow。

说实话，po 主的英文很烂，烂得掉渣了，只有小学5年级的水准（po 主那个时候是四年纪开始学的英语）所以不到逼不得已都不想投靠外国网站，实在是看不到，这搜索不来啊！

我找了足足一天得谷歌，最后通过谷歌找到了 stackoverflow 上有这个相同的问题（我这英文的水平只能通过谷歌使用了，捂脸）

人家大神解答到，原来尼玛的4.0以后的版本又多了一个参数于是乎，再加一个 openFileChooser(ValueCallback<Uri> uploadFile, String acceptType,String capture)方法就可以了。

```
abstract class TestWebChromeClientextends WebChromeClient
{

    private WebChromeClient mWrappedClient;

    protected          TestWebChromeClient(WebChromeClient
wrappedClient)
    {
        mWrappedClient = wrappedClient;
    }

    /** {@inheritDoc} */
    @Override
    public    void    onProgressChanged(WebView    view,int
newProgress)
    {
        mWrappedClient.onProgressChanged(view,
newProgress);
    }

    /** {@inheritDoc} */
    @Override
    public void onReceivedTitle(WebView view, String title)
    {
        mWrappedClient.onReceivedTitle(view, title);
    }

    /** {@inheritDoc} */
    @Override
    public void onReceivedIcon(WebView view, Bitmap icon)
    {
        mWrappedClient.onReceivedIcon(view, icon);
    }

    /** {@inheritDoc} */
    @Override
    public void onReceivedTouchIconUrl(WebView view, String
url,boolean precomposed)
    {
        mWrappedClient.onReceivedTouchIconUrl(view,    url,
precomposed);
    }

    /** {@inheritDoc} */
    @Override
    public      void      onShowCustomView(View      view,
CustomViewCallback callback)
    {
```

```
49          mWrappedClient.onShowCustomView(view, callback);
50      }
51
52      /** {@inheritDoc} */
53      @Override
54      public void onHideCustomView()
55      {
56          mWrappedClient.onHideCustomView();
57      }
58
59      /** {@inheritDoc} */
60      @Override
61      public boolean onCreateWindow(WebView view,boolean
62  dialog,boolean userGesture,
63              Message resultMsg)
64      {
65          return mWrappedClient.onCreateWindow(view, dialog,
66  userGesture, resultMsg);
67      }
68
69      /** {@inheritDoc} */
70      @Override
71      public void onRequestFocus(WebView view)
72      {
73          mWrappedClient.onRequestFocus(view);
74      }
75
76      /** {@inheritDoc} */
77      @Override
78      public void onCloseWindow(WebView window)
79      {
80          mWrappedClient.onCloseWindow(window);
81      }
82
83      /** {@inheritDoc} */
84      @Override
85      public boolean onJsAlert(WebView view, String url, String
86  message, JsResult result)
87      {
88          return mWrappedClient.onJsAlert(view, url, message,
89  result);
90      }
91
92      /** {@inheritDoc} */
93      @Override
94      public boolean onJsConfirm(WebView view, String url,
95  String message, JsResult result)
```

```
96          {
97              return    mWrappedClient.onJsConfirm(view,    url,
98      message, result);
99          }
100
101         /** {@inheritDoc} */
102         @Override
103         public boolean onJsPrompt(WebView view, String url,
104     String message,
105                 String defaultValue, JsPromptResult result)
106         {
107             return    mWrappedClient.onJsPrompt(view,    url,
108     message, defaultValue, result);
109         }
110
111         /** {@inheritDoc} */
112         @Override
113         public boolean onJsBeforeUnload(WebView view, String
114     url, String message,
115                 JsResult result)
116         {
117             return mWrappedClient.onJsBeforeUnload(view, url,
118     message, result);
119         }
120
121         /** {@inheritDoc} */
122         @Override
123         public void onExceededDatabaseQuota(String url, String
124     databaseIdentifier,
125                 long    currentQuota,long    estimatedSize,long
126     totalUsedQuota,
127                 WebStorage.QuotaUpdater quotaUpdater)
128         {
129             mWrappedClient.onExceededDatabaseQuota(url,
130     databaseIdentifier, currentQuota,
131                     estimatedSize,              totalUsedQuota,
132     quotaUpdater);
133         }
134
135         /** {@inheritDoc} */
136         @Override
137         public        void        onReachedMaxAppCacheSize(long
138     spaceNeeded,long totalUsedQuota,
139                 WebStorage.QuotaUpdater quotaUpdater)
140         {
141             mWrappedClient
142                     .onReachedMaxAppCacheSize(spaceNeeded,
```

```
143     totalUsedQuota, quotaUpdater);
144         }
145 
146         /** {@inheritDoc} */
147         @Override
148         public void onGeolocationPermissionsShowPrompt(String
149     origin,
150                 GeolocationPermissions.Callback callback)
151         {
152 
153     mWrappedClient.onGeolocationPermissionsShowPrompt(origin,
154     callback);
155         }
156 
157         /** {@inheritDoc} */
158         @Override
159         public void onGeolocationPermissionsHidePrompt()
160         {
161 
162     mWrappedClient.onGeolocationPermissionsHidePrompt();
163         }
164 
165         /** {@inheritDoc} */
166         @Override
167         public boolean onJsTimeout()
168         {
169             return mWrappedClient.onJsTimeout();
170         }
171 
172         /** {@inheritDoc} */
173         @Override
174         @Deprecated
175         public void onConsoleMessage(String message, int
176     lineNumber, String sourceID)
177         {
178             mWrappedClient.onConsoleMessage(message,
179     lineNumber, sourceID);
180         }
181 
182         /** {@inheritDoc} */
183         @Override
184         public boolean onConsoleMessage(ConsoleMessage
185     consoleMessage)
186         {
187             return
188     mWrappedClient.onConsoleMessage(consoleMessage);
189         }
```

```
190
191         /** {@inheritDoc} */
192         @Override
193         public Bitmap getDefaultVideoPoster()
194         {
195             return mWrappedClient.getDefaultVideoPoster();
196         }
197
198         /** {@inheritDoc} */
199         @Override
200         public View getVideoLoadingProgressView()
201         {
202             return
203     mWrappedClient.getVideoLoadingProgressView();
204         }

        /** {@inheritDoc} */
        @Override
        public  void  getVisitedHistory(ValueCallback<String[]>
    callback)
        {
            mWrappedClient.getVisitedHistory(callback);
        }

        /** {@inheritDoc} */

        public      void      openFileChooser(ValueCallback<Uri>
    uploadFile)
        {
            ((TestWebChromeClient)
    mWrappedClient).openFileChooser(uploadFile);
        }

        /** {@inheritDoc} */

        public      void      openFileChooser(ValueCallback<Uri>
    uploadFile,  String acceptType)
        {
            ((TestWebChromeClient)
    mWrappedClient).openFileChooser(uploadFile,  acceptType);
        }

        /** {@inheritDoc} */

        public      void      openFileChooser(ValueCallback<Uri>
    uploadFile,  String acceptType,
                String capture)
```

```
    {
        ((TestWebChromeClient)
mWrappedClient).openFileChooser(uploadFile, acceptType,
                capture);
    }
}
```

下面我贴上 TestWebChromeClient 的完整代码。

一下是 setWebChromeClient 需要添加的方法。

```
1  public void openFileChooser(ValueCallback<Uri> uploadFile, String acceptType)
2  {
3      openFileChooser(uploadFile);
4  }
5
6  public void openFileChooser(ValueCallback<Uri> uploadFile, String acceptType,
7                              String capture)
8  {
9      openFileChooser(uploadFile);
10 }
```

原文

http://www.cnblogs.com/qinxianyuzou/p/3490243.html?utm_source=tuicool

2013年个人微博推荐技术资料汇总

2013年，过的很充实，生活上如此，技术上亦是。这一年，看了很多的技术资料，技术上也有了很大的提高。而且，本着分享的精神，很多好的技术资料，也都在个人微博@何_登成 上做了推荐。今天，下定决心将整个2013年在微博上推荐的技术资料整理了一下，说真的，写的不少，看的更多。

下面的这些资料，都是精品资料，个人已经看了其中的95%左右，余下未看的，需要找时间看完，已经看过的，也准备找时间多温习几遍，好东西，不怕多看。对于个人来说，这算是一个总结与收藏；对于阅读此博文的朋友来说，也可以各取所需，一起追求技术的进步。

注：资料的组织，先按照领域划分，包括：(Concurrent) Programming、Data Structure & Algorithm、Database（综合、MySQL、Oracle）、Performance、Distributed、OS & Hardware、(New) System、其他 等8个大类。然后针对每一个大类，再按照书籍、博客文章、PPT & PDF 的形式归类组织。

(Concurrent) Programming

书籍

Agner Fog. Optimizing software in C++ An optimization guide for Windows, Linux and Mac

Ulrich Drepper. What Every Programmer Should Know About Memory

博客文章

Bill Liu. 提高软件质量实践——google 篇.

Stan Shebs. GDB

酷壳. Linus：利用二级指针删除单向链表

杨志丰. 定位多线程内存越界问题实践总结

范凯. Web 应用的缓存设计模式

长孙泰. 自旋锁 spinlock 剖析与改进

Paul Hammant. Google’s Scaled Trunk Based Development

Sijin Joseph. Programmer Competency Matrix

Jeff Preshing. preshing on programming

Jeff Darcy. High-Performance Server Architecture

Igor Ostrovsky. Gallery of Processor Cache Effects

John Sladek. Modern Microprocessors - A 90 Minute Guide

Bruce Dawson. Lockless Programming Considerations for Xbox 360 and Microsoft Windows

1024cores. Pointer Packing

Google. Optimal Logging

Martin Thompson. Java Garbage Collection Distilled

何登成. C/C++ Volatile 关键词深度剖析

Herb Sutter. Effective Concurrency: Know When to Use an Active Object Instead of a Mutex

何登成. 并发编程系列之一：锁的意义

Android. SMP Primer for Android

PPT & PDF

淘宝鸣嵩. Treelink 模型预测算法比赛分享.

Google. Automated Locality Optimization Based on the Reuse Distance of String Operations

Paul E. McKenney. Memory Barriers: a Hardware View for Software Hackers

Martin Thompson. Lock-Free Algorithms

Martin Thompson. MYTHBUSTING MODERN HARDWARE TO GAIN “MECHANICAL SYMPATHY”

何登成. CPU Cache and Memory Ordering——并发程序设计入门

Bryan Cantrill. Real-World Concurrency

Paul E. McKenney. Selecting Locking Designs for Parallel Programs

Paul E. McKenney. Selecting Locking Primitives for Parallel Programs

Herb Sutter. atomic Weapons The C++11 Memory Model and Modern Hardware

Herb Sutter. The Free Lunch Is Over A Fundamental Turn Toward Concurrency in Software

Scott Meyers. C++ and the Perils of Double-Checked Locking

David Bacon. The "Double-Checked Locking is Broken" Declaration

Data Structure & Algorithm

Theppitak. An Implementation of Double-Array Trie

BigO. 常用算法和数据结构的复杂度速查表

Sanjay Ghemawat. TCMalloc : Thread-Caching Malloc

Josh Haberman. State of the hash functions, 2012

StackExchange. Core algorithms deployed

usfca. Data Structure Visualizations

Bob Nystrom. Baby's First Garbage Collector

Database

综合

ACM Sigmod. Fifty Years of Databases

何登成. SQL 中的 where 条件，在数据库中提取与应用浅析

Michael J.Franklin. Concurrency Control and Recovery

牛新庄. DB2和 Oracle 的并发控制（锁）比较

PG Selinger. Access path selection in a relational database management system

Michael Stonebraker. Architecture of a Database System

MySQL

书籍

Sasha Pachev. Understanding MySQL Internals

博客文章

Kristian Nielsen. Global transaction ID in MariaDB.

Jeremy Cole. InnoDB: A journey to the core: At the MySQL Conference.

何登成. 从 MySQL Bug#67718浅谈 B+树索引的分裂优化.

InnoDB Team. Repeatable Read Isolation Level in InnoDB - How Consistent Read View Works.

Jeremy Cole. The MySQL “swap insanity” problem and the effects of the NUMA architecture.

张洋. MySQL 索引背后的数据结构及算法原理

InnoDB Team. Introduction to Transaction Locks in InnoDB Storage Engine

何登成. MySQL 加锁处理分析

PPT & PDF

何登成. MySQL 查询优化浅析

MySQL. Understanding and control of MySQL Query Optimizer

Heikki Tuuri. Concurrency Control: How It Really Works

Heikki Tuuri. Crash Recovery and Media Recovery in InnoDB

Calvin Sun. InnoDB: Status,Architecture,and Latest Enhancements

何登成. MySQL 5.6新特性深入剖析——InnoDB 引擎

Peter Zaitsev. INNODB ARCHITECTURE AND PERFORMANCE OPTIMIZATION

MySQL. MySQL Metadata Locking

Effective MySQL. Understanding Tokutek Fractal Tree Indexes

Dimitri KRAVTCHUK. MySQL 5.6 Performance: Tuning and “Best” Practices..

Oracle

书籍

Jonathan Lewis. Oracle Core: Essential Internals for DBAs and Developers.

Jonathan Lewis. Cost-Based Oracle Fundamentals.

Steve Adams. Oracle8i Internal Services for Waits, Latches, Locks, and Memory

博客文章

Jonathan Lewis. Compression in Oracle.

PPT & PDF

何登成. Oracle RAC PCM Cache-Fusion 分析

Performance

书籍

Brendan Gregg. Systems Performance: Enterprise and the Cloud

博客文章

High Scalability. 42 Monster Problems That Attack As Loads Increase

Jeff Dean. Latency numbers every programmer should know

PPT & PDF

Cary Millsap. Thinking Clearly about Performance

Brendan Gregg. Thinking Methodically about Performance

Raj Jain. Operational Laws

何登成. 排队论及其应用浅析

叶正盛. 面向程序员的数据库访问性能优化法则

Distributed

PPT & PDF

Jeff Dean. Large-Scale Data and Computation: Challenges and Opportunities.

Paper Trail. consensus

Mark McKeown. A brief history of Consensus, 2PC and Transaction Commit

Google. Spanner: Google’s Globally-Distributed Database

ImportNew. 经典论文翻译导读之《Finding a needle in Haystack: Facebook’s photo storage》

汪源. AWS 历次事故分析及启示

汪源. 分布式系统设计模式

Ulf Wendel. Data massage! Databases scaled from one to one million nodes

OS & Hardware

书籍

Intel. Intel 64 and IA-32 Architectures Software Developer’ s Manual Volume 3A: System Programming Guide, Part 1

DJ Sorin. A Primer on Memory Consistency and Cache Coherence

博客文章

酷壳. 应该知道的 Linux 技巧.

霸爷. MYSQL 数据库网卡软中断不平衡问题及解决方案.

redhat. Interrupt and Process Binding.

makelinux. Linux Kernel Map.

霸爷. Linux Used 内存到底哪里去了？.

淘宝千石. OOM 原理分析

霸爷. Linux 常用性能调优工具索引

霸爷. Understanding Linux CPU Load 资料汇总

Dan Siemon. QUEUEING IN THE LINUX NETWORK STACK

Sebastian Anthony. How long do hard drives actually live for?

SAE. Linux 下高并发 socket 最大连接数所受的各种限制

PPT & PDF

intel. Understanding the Flash Translation Layer (FTL) Specification

Lanyue Lu. A Study of Linux File System Evolution

Brendan Gregg. Linux Performance Analysis and Tools

John Beckett. NUMA Best Practices for Dell PowerEdge 12th Generation Servers

(New) System

wired. Return of the Borg: How Twitter Rebuilt Google's Secret Weapon

林仕鼎. 系统架构领域的一些学习材料

HS. The Secret To 10 Million Concurrent Connections -The Kernel Is The Problem, Not The Solution

Jeff Dean. Lessons Learned While Building Infrastructure Software at Google

Jay Kreps. The Log: What every software engineer should know about real-time data's unifying abstraction

其他

腾讯科技. 对话另一个世界

汪源. Just Works 的力量——Google 科学家 Jeff Dean 斯坦福大学演讲的启示

熊辉. 为什么人前进的路总是被自己挡住

Google. HOW SEARCH WORKS

wired. If Xerox PARC Invented the PC, Google Invented the Internet

Abel Avram. Are Older Programmers More Knowledgeable?

hingo. 5 years of MySQL

wired. How Three Guys Rebuilt the Foundation of Facebook

36Kr. Google 帝国威武：宕机5分钟，全球网络流量暴跌40%

Quora. What are all the Jeff Dean facts?

图灵社区. 云风：一个编程的自由人（图灵访谈）

何登成. 个人订阅的10佳博客与相关介绍

ExtremeTech. Researchers crack the world's toughest encryption by listening to the tiny sounds made by your computer's CPU

原文 http://hedengcheng.com/?p=828&utm_source=tuicool

AliRedis 单机180w QPS, 8台服务器构建1000w QPS Cache 集群

引言： 如今 redis 凭借其高性能的优势，以及丰富的数据结构作为 cache 已越来越流行，逐步取代了 memcached 等 cache 产品，在 Twitter,新浪微博中广泛使用,阿里巴巴同样如此. redis 已经占据了其不可动摇的地位，然而在实际的生产环境中，redis 也暴露出一些其他问题.如性能瓶颈，内存冗余，运维部署复杂等.本文目的就是分析 redis 现有的问题，以及介绍阿里巴巴技术保障团队对 redis 的改造工作，使其在生产环境中发挥更大的价值.**原生 redis 的瓶颈**:

1. 单进程单线程，无法充分发挥服务器多核 cpu 的性能.
2. 大流量下造成 IO 阻塞. 同样是由于单进程单线程，cpu 在处理业务逻辑的时候，网络 IO 被阻塞住，造成无法处理更多的请求.
3. 维护成本高，如果想要充分发挥服务器的所有资源包括 cpu，网络 io 等，就必须建立多个 instance，但此时不可避免会增加维护成本. 拿24核服务器举例来讲,如果部署24个单机版的 instance,理论上可以实现10w*24core= 240wQPS 的总体性能.但是每个 instance 有各自独立的数据,占用资源如内存也会同比上升,反过来制约一台服务器又未必能支持这么多的 instance. 如果部署24个 Instance 来构成单机集群，虽然可以共享数据，但是因为节点增加，redis 的状态通讯更加频繁和费时,性能也下会降很多. 并且两种方式都意味着要维护24个 Instance，运维成本都会成倍增加.

4. 持久化. redis 提供了两种 save 方式 1)save 触发. 2)bgsave. 当然也可以使用3)aof 来实现持久化，但是这3点都有弊端.

1)save: 由于是单进程单线程，redis 会阻塞住所有请求，来遍历所有 redisDB，把 key-val 写入 dump.rdb. 如果内存数据量过大，会造成短时间几秒到几十秒甚至更长的时间停止服务，这种方案对于 twitter，taobao 等大流量的网站,

显然是不可取的.

2)bgsave: 在触发 bgsave 时, redis 会 fork 自身, child 进程会进入1)的处理方式,这意味着服务器内存要有一半的冗余才可以, 如今内存已变得越来越廉价, 但是对于存储海量数据的情况,内存以及服务器的成本还是不容忽视的.

3)aof: 说到持久化, redis 提供的 aof 算是最完美的方案了, 但是有得必有失, 严重影响性能! 因为 redis 每接收到一条请求, 就要把命令内容完整的写到磁盘文件, 且不说频繁读写会影响磁盘寿命,写磁盘的时间足以拖垮 redis 整体性能 . 当然熟悉 redis 的开发者会想到用 appendfsync 等参数来调整, 但都不是完美.即使使用 SSD，性能也只是略有提升，并且性价比不高。针对以上几种情况, 阿里技术保障团队做了如下优化手段, 其实这不仅仅只是优化, 而更是一种对 redis 的改造.

1. 多线程 master + N*work 工作模式.master 线程负责监听网络事件, 在接收到一个新的连接后, master 会把新的 fd 注册到 worker 的 epoll 事件中, 交由 worker 处理这个 fd 的所有读写事件, 这样 master 线程就可以完全被释放出来接收更多的连接, 同时又不妨碍 worker 处理业务逻辑和 IO 读写.

采用这种 master + N*worker 的网络层事件模型,可以实现 redis 性能的平行扩展.真正的让 redis 在面临高并发请求时可以从容面对.

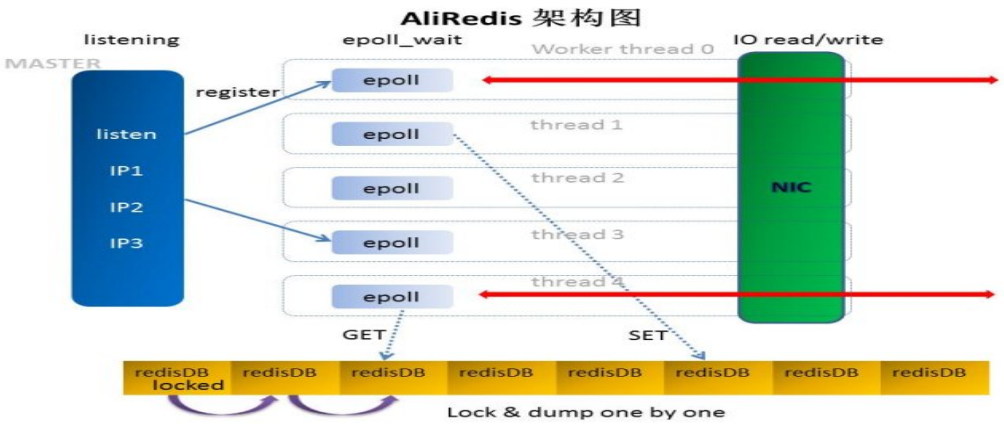
2. 抛弃 save, bgsave, aof 等三种模式.采用 redisDB lock 模式. AliRedis 在数据存储层把多 DB 存储模式转换成 HashDb 存储, 将 key hash 到所有 RedisDB 上, 这样做有一个弊端就是抛弃了 select 命令, 但与此同时会带来一个更大的好处, 可以逐个 DB 持久化而不会影响整个系统, 在做持久化的时候 AliRedis 只需 lock 住 1/N 个 redisDb, 占用1/m 个线程. 在不需要内存冗余的情况下进行持久化, 相比之前提到的弊端, 这种方式可以带来更大的收益, 更丰厚的回报.

3. 重构 hiredis 客户端, 支持 redis-cluster 工作模式, 目前 hiredis 并不支持 redis-cluster 模式, 阿里技术保障团队对 hiredis 进行重构,使之支持 redis-cluster.

4. 优化 jemalloc, 采用大内存页. Redis 在使用内存方面可谓苛刻至极, 压缩, string 转 number 等, 能省就省, 但是在实际生产环境中, 为了追求性能, 对于内存的使用可以适度(不至于如bgsave般浪费)通融处理, 因此AliRedis对jemalloc做了微调, 通过调整 pagesize 来让一次 je_malloc 分配更多 run 空间来储备更多的用户态可用内存, 同时可以减轻换页表的负载, 降低 user sys 的切换频率, 提

高申请内存的性能，对 jemalloc 有兴趣的开发者可以参考 jemalloc 源码中的 bin, run, chunk 数据结构进行分析.

通过如上改造，redis 可以充分发挥服务器多核的优势，以及网络 IO 复用，特别是节省运维成本，每台服务器只需维护一个 AliRedis 实例.



测试环境

CPU: Intel Xeon E5-2630 2.3GHz, *2

KERNEL: 3.2.0

GCC: 4.4.6

Jemalloc: 3.2.0

NIC: Intel 82599EB **测试数据**

AliRedis 单机版性能数据

thread: 1 2 4 8 12 16 20 24

set/get=1:1 83182 162214 301552 605817 876656 1173748 1551113 1800878

AliRedis 集群

8个节点，20台客户端，配置相同，cpu 全部打满.

set/get=1:1 qps: 10,344,877**结论**单机版 AliRedis 可以实现24核跑满180wQPS 性能.集群版可以实现8台服务器支撑1000wQPS 的数据请求.

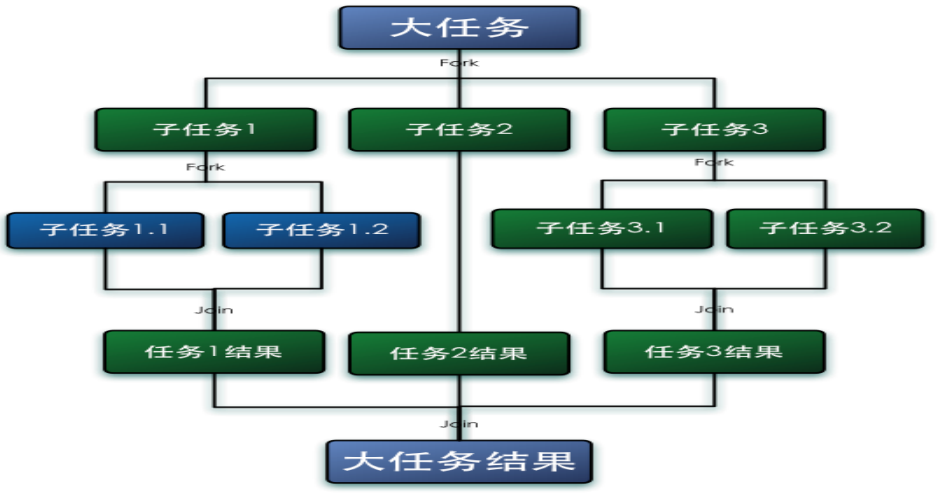
原文 http://blog..sinacom.cn/s/blog_e59371cc0101br74.html

聊聊并发（八）——Fork/Join 框架介绍

1. 什么是 Fork/Join 框架

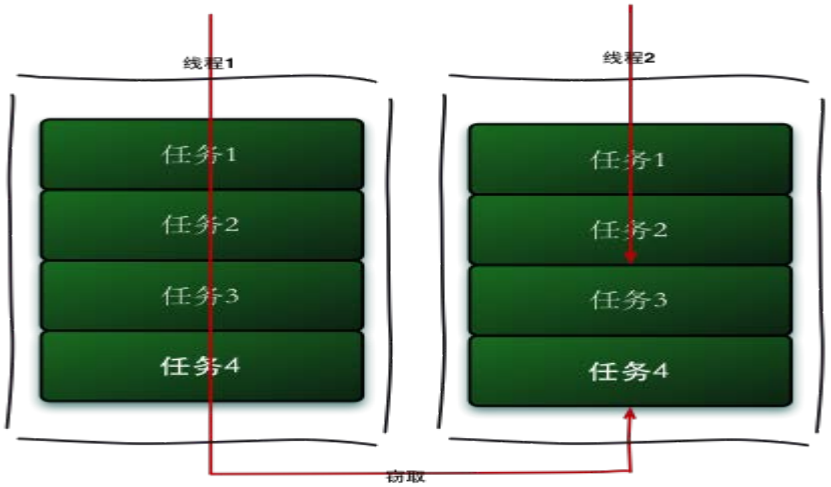
Fork/Join 框架是 Java7提供了的一个用于并行执行任务的框架， 是一个把大任务分割成若干个小任务，最终汇总每个小任务结果后得到大任务结果的框架。

我们再通过 Fork 和 Join 这两个单词来理解下 Fork/Join 框架，Fork 就是把一个大任务切分为若干子任务并行的执行，Join 就是合并这些子任务的执行结果，最后得到这个大任务的结果。比如计算1+2+。。＋10000，可以分割成10个子任务，每个子任务分别对1000个数进行求和，最终汇总这10个子任务的结果。Fork/Join 的运行流程图如下：



2. 工作窃取算法

工作窃取（work-stealing）算法是指某个线程从其他队列里窃取任务来执行。工作窃取的运行流程图如下：



那么为什么需要使用工作窃取算法呢？假如我们需要做一个比较大的任务，我们可以把这个任务分割为若干互不依赖的子任务，为了减少线程间的竞争，于是把这些子任务分别放到不同的队列里，并为每个队列创建一个单独的线程来执行队列里的任务，线程和队列一一对应，比如 A 线程负责处理 A 队列里的任务。但是有的线程会先把自己队列里的任务干完，而其他线程对应的队列里还有任务等待处理。干完活的线程与其等着，不如去帮其他线程干活，于是它就去其他线程的队列里窃取一个任务来执行。而在这时它们会访问同一个队列，所以为了减少窃取任务线程和被窃取任务线程之间的竞争，通常会使用双端队列，被窃取任务线程永远从双端队列的头部拿任务执行，而窃取任务的线程永远从双端队列的尾部拿任务执行。

工作窃取算法的优点是充分利用线程进行并行计算，并减少了线程间的竞争，其缺点是在某些情况下还是存在竞争，比如双端队列里只有一个任务时。并且消耗了更多的系统资源，比如创建多个线程和多个双端队列。

3. Fork/Join 框架的介绍

我们已经很清楚 Fork/Join 框架的需求了，那么我们可以思考一下，如果让我们来设计一个 Fork/Join 框架，该如何设计？这个思考有助于你理解 Fork/Join 框架的设计。

第一步分割任务。首先我们需要有一个 fork 类来把大任务分割成子任务，有可能子任务还是很大，所以还需要不停的分割，直到分割出的子任务足够小。

第二步执行任务并合并结果。分割的子任务分别放在双端队列里，然后几个启动线程分别从双端队列里获取任务执行。子任务执行完的结果都统一放在一个队列里，启动一个线程从队列里拿数据，然后合并这些数据。

Fork/Join 使用两个类来完成以上两件事情：

- ForkJoinTask：我们要使用 ForkJoin 框架，必须首先创建一个 ForkJoin 任务。它提供在任务中执行 fork()和 join()操作的机制，通常情况下我们不需要直接继承 ForkJoinTask 类，而只需要继承它的子类，Fork/Join 框架提供了以下两个子类：
    - RecursiveAction：用于没有返回结果的任务。
    - RecursiveTask ：用于有返回结果的任务。
- ForkJoinPool ：ForkJoinTask 需要通过 ForkJoinPool 来执行，任务分割出的子任务会添加到当前工作线程所维护的双端队列中，进入队列的头部。当一个工作线程的队列里暂时没有任务时，它会随机从其他工作线程的队列的尾部获取一个任务。

4. 使用 Fork/Join 框架

让我们通过一个简单的需求来使用下 Fork / Join 框架，需求是：计算1+2+3+4的结果。

使用 Fork / Join 框架首先要考虑到的是如何分割任务，如果我们希望每个子任务最多执行两个数的相加，那么我们设置分割的阈值是2，由于是4个数字相加，所以 Fork / Join 框架会把这个任务 fork 成两个子任务，子任务一负责计算1+2，子任务二负责计算3+4，然后再 join 两个子任务的结果。

因为是有结果的任务，所以必须继承 RecursiveTask，实现代码如下：

```
package fj;

import java.util.concurrent.ExecutionException;
import java.util.concurrent.ForkJoinPool;
import java.util.concurrent.Future;
import java.util.concurrent.RecursiveTask;

public class CountTask extends RecursiveTask<Integer> {

	private static final int THRESHOLD = 2;// 阈值
	private int start;
	private int end;

	public CountTask(int start, int end) {
		this.start = start;
		this.end = end;
	}
	@Override
	protected Integer compute() {
		int sum = 0;

		// 如果任务足够小就计算任务
		boolean canCompute = (end - start) <= THRESHOLD;
		if (canCompute) {
			for (int i = start; i <= end; i++) {
				sum += i;
			}
		} else {
			// 如果任务大于阈值，就分裂成两个子任务计算
			int middle = (start + end) / 2;
			CountTask leftTask = new CountTask(start, middle);
			CountTask rightTask = new CountTask(middle + 1, end);
			//执行子任务
			leftTask.fork();
			rightTask.fork();
			//等待子任务执行完，并得到其结果
```

```
                int leftResult=leftTask.join();
                int rightResult=rightTask.join();
                //合并子任务
                sum = leftResult  + rightResult;
            }
            return sum;
        }

        public static void main(String[] args) {
            ForkJoinPool forkJoinPool = new ForkJoinPool();
            // 生成一个计算任务，负责计算1+2+3+4
            CountTask task = new CountTask(1, 4);
            // 执行一个任务
            Future<Integer> result = forkJoinPool.submit(task);
            try {
                System.out.println(result.get());
            } catch (InterruptedException e) {
            } catch (ExecutionException e) {
            }
        }
}
```

通过这个例子让我们再来进一步了解 ForkJoinTask，ForkJoinTask 与一般的任务的主要区别在于它需要实现 compute 方法，在这个方法里，首先需要判断任务是否足够小，如果足够小就直接执行任务。如果不足够小，就必须分割成两个子任务，每个子任务在调用 fork 方法时，又会进入 compute 方法，看看当前子任务是否需要继续分割成孙任务，如果不需要继续分割，则执行当前子任务并返回结果。使用 join 方法会等待子任务执行完并得到其结果。

5. Fork/Join 框架的异常处理

ForkJoinTask 在执行的时候可能会抛出异常，但是我们没办法在主线程里直接捕获异常，所以 ForkJoinTask 提供了 isCompletedAbnormally()方法来检查任务是否已经抛出异常或已经被取消了，并且可以通过 ForkJoinTask 的 getException 方法获取异常。使用如下代码：

```
if(task.isCompletedAbnormally())

{

    System.out.println(task.getException());

}
```

getException 方法返回 Throwable 对象，如果任务被取消了则返回 CancellationException。如果任务没有完成或者没有抛出异常则返回 null。

6. Fork/Join 框架的实现原理

ForkJoinPool 由 ForkJoinTask 数组和 ForkJoinWorkerThread 数组组成，ForkJoinTask 数组负责存放程序提交给 ForkJoinPool 的任务，而 ForkJoinWorkerThread 数组负责执行这些任务。

ForkJoinTask 的 fork 方法实现原理。当我们调用 ForkJoinTask 的 fork 方法时，程序会调用 ForkJoinWorkerThread 的 pushTask 方法异步的执行这个任务，然后立即返回结果。代码如下：

```
public final ForkJoinTaskfork() { ((ForkJoinWorkerThread)
Thread.currentThread()) .pushTask(this); return this; }
```

pushTask 方法把当前任务存放在 ForkJoinTask 数组 queue 里。然后再调用 ForkJoinPool 的 signalWork()方法唤醒或创建一个工作线程来执行任务。代码如下：

```
final void pushTask(ForkJoinTask t) {

        ForkJoinTask[] q; int s, m;

        if ((q = queue) != null) {    // ignore if queue removed

            long u = (((s = queueTop) & (m = q.length - 1)) << ASHIFT) +
ABASE;

            UNSAFE.putOrderedObject(q, u, t);

            queueTop = s + 1;         // or use putOrderedInt

            if ((s -= queueBase) <= 2)

                pool.signalWork();

    else if (s == m)

                growQueue();

        }

    }
```

ForkJoinTask 的 join 方法实现原理。Join 方法的主要作用是阻塞当前线程并等待获取结果。让我们一起看看 ForkJoinTask 的 join 方法的实现，代码如下：

```
public final V join() {
        if (doJoin() != NORMAL)
            return reportResult();
        else
            return getRawResult();
}
private V reportResult() {
        int s; Throwable ex;
        if ((s = status) == CANCELLED)
            throw new CancellationException();
if (s == EXCEPTIONAL && (ex = getThrowableException()) != null)
            UNSAFE.throwException(ex);
        return getRawResult();
}
```

首先，它调用了 doJoin()方法，通过 doJoin()方法得到当前任务的状态来判断返回什么结果,任务状态有四种:已完成(NORMAL),被取消(CANCELLED),信号(SIGNAL)和出现异常（EXCEPTIONAL)。

- 如果任务状态是已完成，则直接返回任务结果。
- 如果任务状态是被取消，则直接抛出 CancellationException。
- 如果任务状态是抛出异常，则直接抛出对应的异常。

让我们再来分析下 doJoin()方法的实现代码：

```
private int doJoin() {
        Thread t; ForkJoinWorkerThread w; int s; boolean completed;
```

```
        if ((t = Thread.currentThread()) instanceof ForkJoinWorkerThread)
{
            if ((s = status) < 0)
 return s;
            if ((w = (ForkJoinWorkerThread)t).unpushTask(this)) {
                try {
                    completed = exec();
                } catch (Throwable rex) {
                    return setExceptionalCompletion(rex);
                }
                if (completed)
                    return setCompletion(NORMAL);
            }
            return w.joinTask(this);
        }
        else
            return externalAwaitDone();
    }
```

在 doJoin()方法里，首先通过查看任务的状态，看任务是否已经执行完了，如果执行完了，则直接返回任务状态，如果没有执行完，则从任务数组里取出任务并执行。如果任务顺利执行完成了，则设置任务状态为 NORMAL，如果出现异常，则纪录异常，并将任务状态设置为 EXCEPTIONAL。

原文
http://www.infoq.com/cn/articles/fork-join-introduction?utm_source=tuicool

PostgreSQL 配置优化

硬件和系统配置

操作系统 Ubuntu13.04
系统位数 64
CPU　　Intel(R) Core(TM)2 Duo CPU
内存　　4G
硬盘　　Seagate ST2000DM001-1CH164
测试工具 PostgreSQL-9.1.11

测试工具

工具名称　　pgbench
数据量　　200W（整个数据库大小约为300M）
模拟客户端数 4
线程数　　4
测试时间　　60秒

- 准备命令：pgbench -i -s 20 pgbenchdb
- 测试命令：pgbench -r -j4 -c4 -T60 testdb

配置文件

默认的配置配置文件是保存在/etc/postgresql/VERSION/main 目录下的 postgresql.conf 文件

- 如果想查看参数修改是否生效，可以用 psql 连接到数据库后，用<show 选项名> 来查看。
- 如果要修改 shared_buffers， 在 ubuntu 下可能需要执行命令<sysctl -w>Managing Kernel Resources

主要选项

选项	默认值	说明	是否优化	原因

max_connections	100	允许客户端连接的最大数目	否	因为在测试的过程中，100个连接已经足够
fsync	on	强制把数据同步更新到磁盘	是	因为系统的 IO 压力很大，为了更好的测试其他配置的影响，把改参数改为 off
shared_buffers	24MB	决定有多少内存可以被 PostgreSQL 用于缓存数据（推荐内存的1/4）	是	在 IO 压力很大的情况下，提高该值可以减少 IO
work_mem	1MB	使内部排序和一些复杂的查询都在这个 buffer 中完成	是	有助提高排序等操作的速度，并且减低 IO
effective_cache_size	128MB	优化器假设一个查询可以用的最大内存，和 shared_buffers 无关（推荐内存的1/2）	是	设置稍大，优化器更倾向使用索引扫描而不是顺序扫描
maintenance_work_mem	16MB	这里定义的内存只是被 VACUUM 等耗费资源较多的命令调用时使用	是	把该值调大，能加快命令的执行
wal_buffer	768kB	日志缓存区的大小	是	可以降低 IO，如果遇上比较多的并发短事务，应该和 commit_delay 一起用
checkpoint_segments	3	设置 wal log 的最大数量数（一个 log 的大小为16M）	是	默认的48M的缓存是一个严重的瓶颈，基本上都要设置为10以上
checkpoint_completion_target	0.5	表示 checkpoint 的完成时间要在两个 checkpoint 间隔时间的 N%内完成	是	能降低平均写入的开销

commit_delay	0	事务提交后，日志写到 wal log 上到 wal_buffer 写入到磁盘的时间间隔。需要配合 commit_sibling	是	能够一次写入多个事务，减少 IO，提高性能
commit_siblings	5	设置触发 commit_delay 的并发事务数，根据并发事务多少来配置	是	减少 IO，提高性能

测试数据

- 测试的数据是运行3次，取平均值。
- 关闭 fsync 是为了更好的体现出其他参数对 PostgreSQL 的影响。

参数	修改值	事务总数	tps(包括建立连接)	tps(不包括建立连接)
默认设置		8464	140.999792	141.016182
fsync	off	92571	1479.969755	1480.163355
shared_buffers	1GB	100055	1635.759275	1635.977823
work_mem	10MB	101209	1665.804812	1666.04082
effective_cache_size	2GB	98209	1636.733152	1636.970271
maintenance_work_mem	512MB	92930	1548.029233	1548.223108
checkpoint_segments	32	195982	3265.995	3266.471064
checkpoint_completion_target	0.9	194390	3239.406493	3239.842596
wal_buffer	8MB	198639	3310.241458	3310.724067
恢复 fsync	off	11157	185.883542	185.909849
commit_delay && commit_siblings	10 && 4	11229	187.103538	187.131747

总结

	事务总数	tps(包括建立连接)	tps(不包括建立连接)
优化前	8464	140.999792	141.016182
优化后(fsync=on)	11229	187.103538	187.131747
优化后(fsync=off)	198639	3310.241458	3310.724067

在 fsync 打开的情况下，优化后性能能够提升30%左右。因为有部分优化选项在默认的 SQL 测试语句中没有体现出它的优势，如果到实际测试中，提升应该不止30%。测试的过程中，主要的瓶颈就在系统的 IO，如果需要减少 IO 的负荷，最直接的方法就是把 fsync 关闭，但是这样就会在掉电的情况下，可能会丢失部分数据。

原文

http://blog.csdn.net/kyle__shaw/article/details/17576259?utm_source=tuicool

数据库优化的最佳实践

在许多很好的例子，技术和方法被世界上最好的数据库性能专家所高荐。我们将讨论提高数据库性能的最常用的方法，而不是评论或建议任何特定的工具或技术。

1）谨慎而有效地使用索引

选择合理的索引（前缀性及可选性）、删除没有用的索引。

2）使用规范化，但不要使用过头

规范化(至少是第三范式)是一个易于理解且标准的方法。然而，在有些情况下，你可能希望违反这些规则。查询表通常是规范化的产物，也就是说，你创建了一个特殊的表，这个表包含了在其他表中被频繁使用的相关信息的列表。然而，当使用那些经常被访问且分布有限(仅有或有限的行数拥有小值）的查找表时，会使系统性能降低。在这种情况下，每次你使用查询信息，它们必须使用 join 以获取完整数据。join 的开销很大，而且频繁访问会使开销随着时间逐渐增加。为了减少这种潜在的性能问题，可以使用枚举字段存储数据，而不是使用查找表存储数据。例如，可以使用枚举字段存储头发彩色值，而不是创建表来存储头发颜色值，这样还可以避免使用 join.

3）使用正确的存储引擎

mysql 的最强大的功能之一是它支持不同的存储引擎，存储引擎管理如何存储和恢复数据。mysql 支持多个存储引擎，每个存储引擎具有独特的功能和用途，可以使数据库设计通过使用最合适他们的应用程序的存储引擎来改善数据库系统的性能。例如，如果有一个这样的环境：使用事务控制高度活跃的数据库，请选择一个合适这个情况的存储引擎（mysql 的有些存储引擎不支持事务），你还可能会发现这样的视图和表，它们常常被查询但是几乎不被更新(例如查找表)，在这种情况下，你可能希望使用存储引擎将这些数据存储在内存中，以便快速访问它们。

InnoDB 存储引擎支持事务，在需要事务支持时，通常应该选择这个存储引擎，它是 Mysql 中目前唯一事务性的引擎。很多第三方存储引擎支持事务，但是仅有 InnoDB 有"开包即用"选项。有趣的是，InnoDB 中所有的索引都是 B-trees，在这个 B 树中索引记录被存储在树的叶子项， InnoDB 适用于高性能和事务处理环境。

MyISAM 存储引擎是 Mysql 默认引擎，如果你在 create 语句中省略了 engine 选项，那么默认使用这个引擎。MyISAM 经常在数据仓库、电子商务和企业应用中使用。MyISAM 使用高级缓存和索引机制提高数据检索速度，另外，当各种应用程序需要快速检索数据而不需要事务时，MyISAM 将是很好的选择。

Blackhole 存储引擎是非常有趣的，它并不存储任何东西。实际上，正如它的名字所言-存储进去的数据永远还会返回。Blackhole 存储引擎有个特殊的用途，如果启用了二进制日志，SQL 命令将被写入这个日志，这时，Blackhole 存储引擎被当做复制拓扑中的中继代理使用。

Memory 存储引擎(有时被称为 HEAP)是内存中的存储器，它使用哈希机制频繁检索被使用过的数据，这样可以更快地检索，它访问数据的方式与其他存储引擎类似，但是数据存储在内存中，并且只在 mysql 会话有效。当关机时，这些数据被刷新并删除掉。Memory 存储引擎通常用于以下情况：静态数据被频繁使用且很少被改变（如查找表).

4）通过 Query Cache 使用视图来加速结果

5）使用约束

6）使用 explain、analyze、optimize

这些工具在诊断和调优时很重要,在不发生错误的前提下经常使用它们,但是请小心使用。具体来说，当 analyze、optimize 有意义且不是作为定期的预定的事件时使用它们。我们发现有些系统管理员晚上使用这些命令，但是一般情况下，这样做是不值得的，并且会产生不必要的表副本。显然,强制系统定期复制数据浪费时间，并会导致操作过程中的访问有限。

原文

http://www.blogjava.net/qileilove/archive/2013/12/24/407960.html?utm_source=tuicool

分布式存储推荐论文

The Google File System. Sanjay Ghemawat, Howard Gobioff, and Shun-Tak Leung。基于普通服务器构建超大规模文件系统的典型案例，简单实用， 是 google 的重要基础设施，大数据的基石， 主要面向大文件和批处理系统。 主要技术特点包括：假设硬件故障是常态， 64MB 大块， 单 Master 设计，Lease/链式复制， 重点支持追加写。

Bigtable: A Distributed Storage System for Structured Data. Fay Chang, Jeffrey Dean, Sanjay Ghemawat, et. 支持 PB 数据量级的多维非关系型大表， 在 google 内部应用广泛，大数据的奠基作品之一 ， Hbase 就是仿照 BigTable 设计。Bigtable 的主要技术特点包括： 底层存储使用 GFS， 使用非原地更新技术实现数据修改， 采用 range 分区等。

Spanner: Google's Globally-Distributed Database. James C. Corbett, Jeffrey

Dean, et. 第一个用于大规模产品的， 高可用，跨数据中心，支持事务的分布式数据库。 主要技术特点包括， 基于 GPS 和原子钟全球同步时间 TrueTime， Paxo，多版本事务。

PacificA: Replication in Log-Based Distributed Storage Systems. Wei Lin, Mao Yang, et. 支持强一致的 log-based 存储的主从复制协议， 具有实用性， 能加深对主从强一致复制的理解程度。 技术特点： 支持强一致主从复制协议， 允许多种存储实现， 分布式的故障检测/Lease/成员管理方法。

Object Storage on CRAQ, High-throughput chain replication for read-mostly workloads. Jeff Terrace and Michael J. Freedman. 支持强一直的链式复制方法， 支持从多个副本读取数据。

Ceph: Reliable, Scalable, and High-Performance Distributed Storage. Sage A. Weil. 功能强大的开源海量存储系统， 支持文件系统、块设备、以及 S3接口。主要技术特色： CRUSH 数据对象定位算法， 基于动态子树的文件系统元数据管理。

Finding a needle in Haystack: Facebook’s photo storage. Doug Beaver, Sanjeev Kumar, Harry C. Li, Jason Sobel, Peter Vajgel. Facebook 分布式 Blob 存储， 主要用于存储图片。 主要技术特色： 小文件合并大文件， 小文件元数据放在内存读写只需一次 IO。

Windows Azure Storage: A Highly Available Cloud Storage Service with Strong Consistency. Brad Calder, Ju Wang, Aaron Ogus, Niranjan Nilakantan, et. 微软的分布式存储平台， 支持 S3对象存储、表格、队列等数据模型。 主要技术特点： 类似 BigTable Stream/Partition 两层设计；写错（写满）就封存 Extent，使得副本字节一致， 简化了选主和恢复操作； 将 S3对象存储、表格、队列、块设备等融入到统一的底层存储架构中。

The Chubby lock service for loosely-coupled distributed systems. Mike Burrows. 高可用、可靠的分布式锁服务，可用于实现选主加锁等功能，ZooKeeper 就是根据此论文实现。 主要技术特点： 将 paxo 协议封装成文件系统接口， 高可用、高可靠，但是不保证性能。

Paxos Made Live - An Engineering Perspective. Tushar Chandra, Robert Griesemer, Joshua Redstone. Paxo 在 chubby 中的工程应用论文， 是理解 Paxo 协议， 及其应用场景的必备论文。 主要技术特点： paxo 协议， replicated log，multi-paxo。

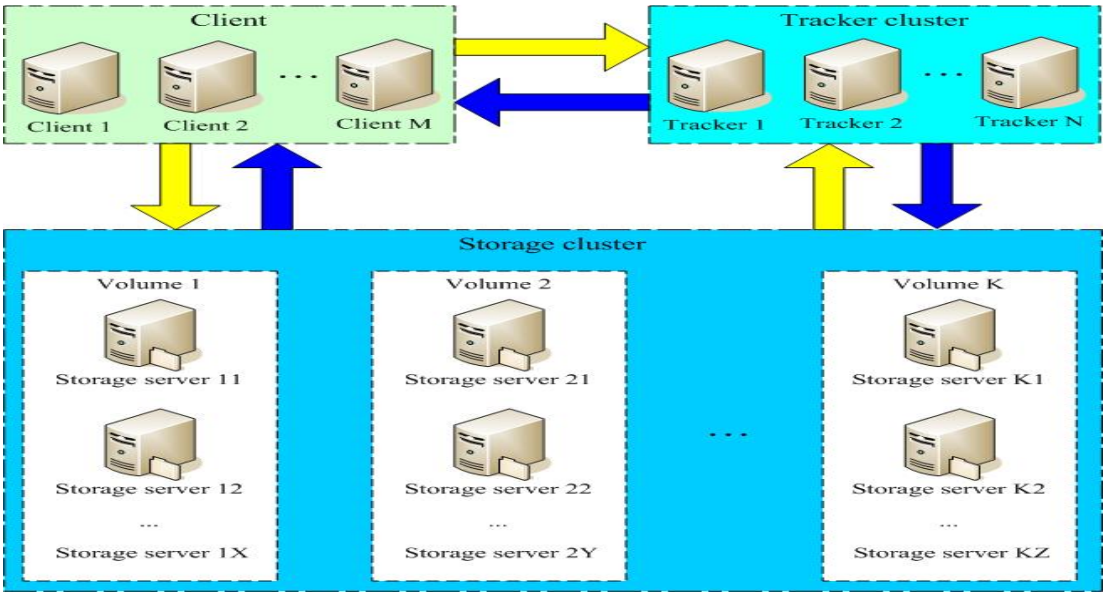
Dynamo: Amazon’s Highly Available Key-Value Store。 Giuseppe DeCandia, Deniz Hastorun, Madan Jampani, et. 高可用的 kv 系统， 主要技术特点：综和运用一致性哈希，vector clock， 最终一致性构建一个高可用的 kv 系统， 可应用于 amazon 购物车场景。

原文

http://www.bitstech.net/2013/12/27/distributed-storage-papers/?utm_source=tuicool

分布式文件系统 FastDFS 设计原理

FastDFS 是一个开源的轻量级分布式文件系统，由跟踪服务器（tracker server）、存储服务器（storage server）和客户端（client）三个部分组成，主要解决了海量数据存储问题，特别适合以中小文件（建议范围：4KB < file_size <500MB）为载体的在线服务。



Storage server

Storage server（后简称 storage）以组（卷，group 或 volume）为单位组织，一个 group 内包含多台 storage 机器，数据互为备份，存储空间以 group 内容量最小的 storage 为准，所以建议 group 内的多个 storage 尽量配置相同，以免造成存储空间的浪费。

以 group 为单位组织存储能方便的进行应用隔离、负载均衡、副本数定制（group 内 storage server 数量即为该 group 的副本数），比如将不同应用数据存到不同的 group 就能隔离应用数据，同时还可根据应用的访问特性来将应用分配到不同的 group 来做负载均衡；缺点是 group 的容量受单机存储容量的限制，同时当 group 内有机器坏掉时，数据恢复只能依赖 group 内地其他机器，使得恢复时间会很长。

group 内每个 storage 的存储依赖于本地文件系统，storage 可配置多个数据存储目录，比如有10块磁盘，分别挂载在/data/disk1-/data/disk10，则可将这10个目录都配置为 storage 的数据存储目录。

storage 接受到写文件请求时，会根据配置好的规则（后面会介绍），选择其中一个存储目录来存储文件。为了避免单个目录下的文件数太多，在 storage 第一次启

动时，会在每个数据存储目录里创建2级子目录，每级256个，总共65536个文件，新写的文件会以 hash 的方式被路由到其中某个子目录下，然后将文件数据直接作为一个本地文件存储到该目录中。

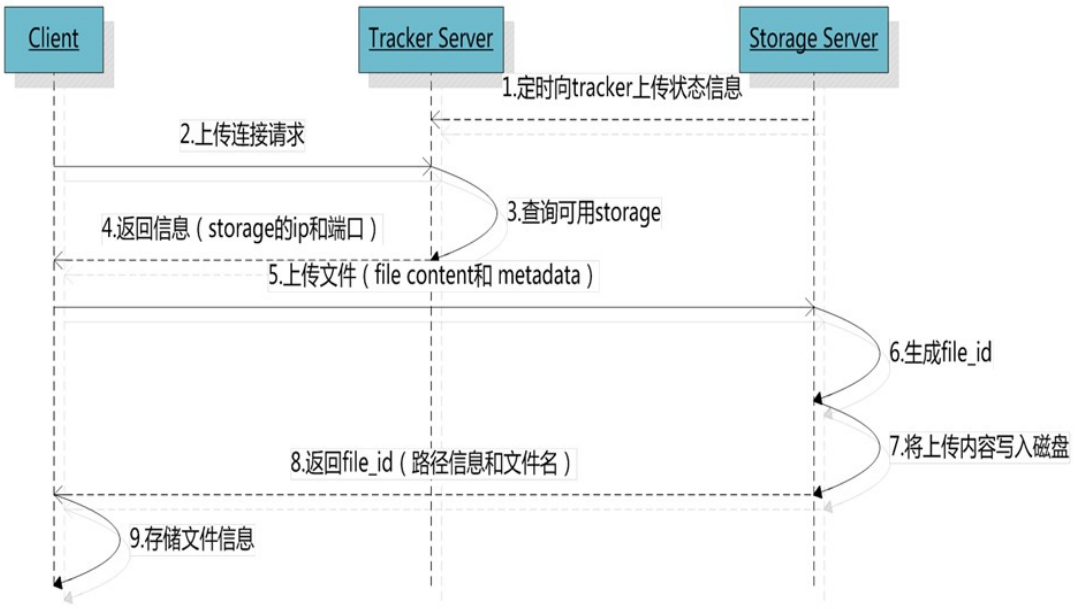
Tracker server

Tracker 是 FastDFS 的协调者，负责管理所有的 storage server 和 group，每个 storage 在启动后会连接 Tracker，告知自己所属的 group 等信息，并保持周期性的心跳，tracker 根据 storage 的心跳信息，建立 group==>[storage server list] 的映射表。

Tracker 需要管理的元信息很少，会全部存储在内存中；另外 tracker 上的元信息都是由 storage 汇报的信息生成的,本身不需要持久化任何数据,这样使得 tracker 非常容易扩展，直接增加 tracker 机器即可扩展为 tracker cluster 来服务，cluster 里每个 tracker 之间是完全对等的，所有的 tracker 都接受 stroage 的心跳信息，生成元数据信息来提供读写服务。

Upload file

FastDFS 向使用者提供基本文件访问接口,比如 upload、download、append、delete 等，以客户端库的方式提供给用户使用。



选择 tracker server

当集群中不止一个 tracker server 时，由于 tracker 之间是完全对等的关系，客户端在 upload 文件时可以任意选择一个 trakcer。

选择存储的 group

当 tracker 接收到 upload file 的请求时，会为该文件分配一个可以存储该文件的 group，支持如下选择 group 的规则：

1. Round robin，所有的 group 间轮询
2. Specified group，指定某一个确定的 group
3. Load balance，剩余存储空间多多 group 优先

选择 storage server

当选定 group 后，tracker 会在 group 内选择一个 storage server 给客户端，支持如下选择 storage 的规则：

1. Round robin，在 group 内的所有 storage 间轮询
2. First server ordered by ip，按 ip 排序
3. First server ordered by priority，按优先级排序（优先级在 storage 上配置）

选择 storage path

当分配好 storage server 后，客户端将向 storage 发送写文件请求，storage 将会为文件分配一个数据存储目录，支持如下规则：

1. Round robin，多个存储目录间轮询
2. 剩余存储空间最多的优先

生成 Fileid

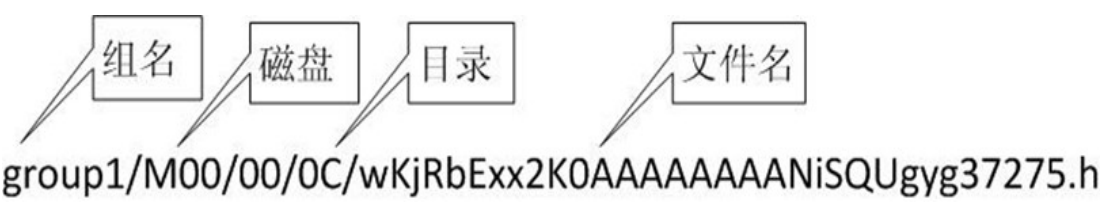
选定存储目录之后，storage 会为文件生一个 Fileid，由 storage server ip、文件创建时间、文件大小、文件 crc32和一个随机数拼接而成，然后将这个二进制串进行 base64编码，转换为可打印的字符串。

选择两级目录

当选定存储目录之后，storage 会为文件分配一个 fileid，每个存储目录下有两级 256*256的子目录，storage 会按文件 fileid 进行两次 hash（猜测），路由到其中一个子目录，然后将文件以 fileid 为文件名存储到该子目录下。

生成文件名

当文件存储到某个子目录后，即认为该文件存储成功，接下来会为该文件生成一个文件名，文件名由 group、存储目录、两级子目录、fileid、文件后缀名（由客户端指定，主要用于区分文件类型）拼接而成。



文件同步

写文件时，客户端将文件写至 group 内一个 storage server 即认为写文件成功，storage server 写完文件后，会由后台线程将文件同步至同 group 内其他的 storage server。

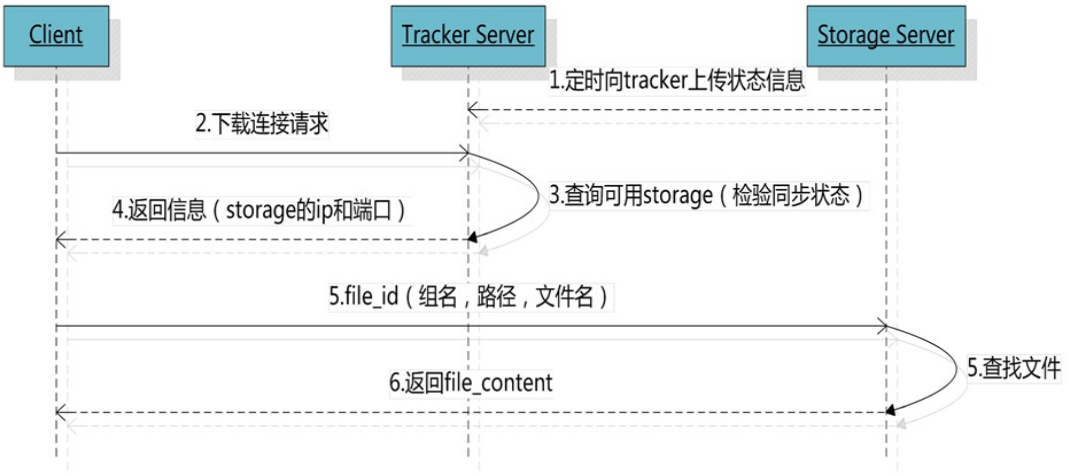
每个 storage 写文件后，同时会写一份 binlog，binlog 里不包含文件数据，只包含文件名等元信息，这份 binlog 用于后台同步，storage 会记录向 group 内其他 storage 同步的进度，以便重启后能接上次的进度继续同步；进度以时间戳的方式进行记录，所以最好能保证集群内所有 server 的时钟保持同步。

storage 的同步进度会作为元数据的一部分汇报到 tracker 上，tracke 在选择读 storage 的时候会以同步进度作为参考。

比如一个 group 内有 A、B、C 三个 storage server，A 向 C 同步到进度为 T1（T1 以前写的文件都已经同步到 B 上了），B 向 C 同步到时间戳为 T2（T2 > T1），tracker 接收到这些同步进度信息时，就会进行整理，将最小的那个做为 C 的同步时间戳，本例中 T1即为 C 的同步时间戳为 T1（即所有 T1以前写的数据都已经同步到 C 上了）；同理，根据上述规则，tracker 会为 A、B 生成一个同步时间戳。

Download file

客户端 upload file 成功后，会拿到一个 storage 生成的文件名，接下来客户端根据这个文件名即可访问到该文件。



跟 upload file 一样，在 download file 时客户端可以选择任意 tracker server。

tracker 发送 download 请求给某个 tracker，必须带上文件名信息，tracke 从文件名中解析出文件的 group、大小、创建时间等信息，然后为该请求选择一个 storage 用来服务读请求。由于 group 内的文件同步时在后台异步进行的，所以有可能出现在读到时候，文件还没有同步到某些 storage server 上，为了尽量避免

访问到这样的 storage，tracker 按照如下规则选择 group 内可读的 storage。

1. 该文件上传到的源头 storage
   - 源头 storage 只要存活着，肯定包含这个文件，源头的地址被编码在文件名中。
2. 文件创建时间戳==storage 被同步到的时间戳 且（当前时间-文件创建时间戳）> 文件同步最大时间（如5分钟）
   - 文件创建后，认为经过最大同步时间后，肯定已经同步到其他 storage 了。
3. 文件创建时间戳 < storage 被同步到的时间戳。
   - 同步时间戳之前的文件确定已经同步了
4. （当前时间-文件创建时间戳）> 同步延迟阀值（如一天）。
   - 经过同步延迟阈值时间，认为文件肯定已经同步了。

小文件合并存储

将小文件合并存储主要解决如下几个问题：

1. 本地文件系统 inode 数量有限，从而存储的小文件数量也就受到限制。
2. 多级目录+目录里很多文件，导致访问文件的开销很大（可能导致很多次 IO）
3. 按小文件存储，备份与恢复的效率低

FastDFS 在 V3.0版本里引入小文件合并存储的机制，可将多个小文件存储到一个大的文件（trunk file），为了支持这个机制，FastDFS 生成的文件 fileid 需要额外增加16个字节

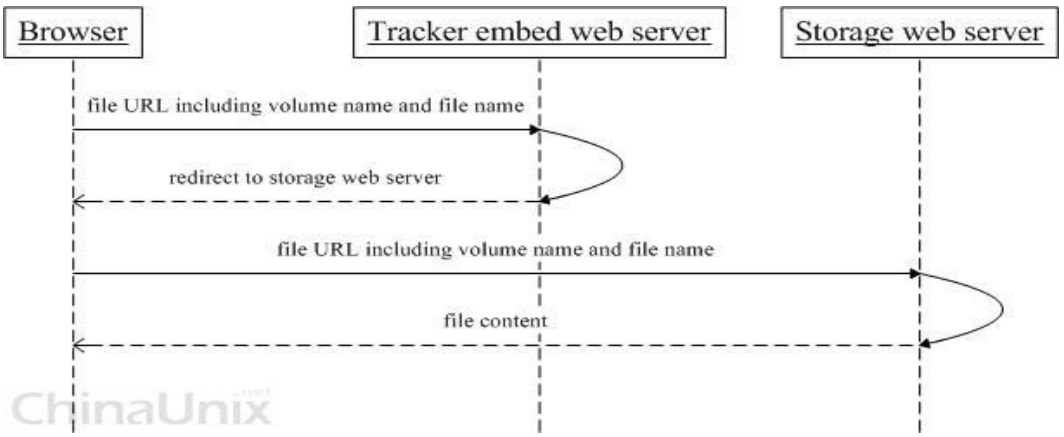
1. trunk file id
2. 文件在 trunk file 内部的 offset
3. 文件占用的存储空间大小 （字节对齐及删除空间复用，文件占用存储空间>=文件大小）

每个 trunk file 由一个 id 唯一标识，trunk file 由 group 内的 trunk server 负责创建（trunk server 是 tracker 选出来的），并同步到 group 内其他的 storage，文件存储合并存储到 trunk file 后，根据其 offset 就能从 trunk file 读取到文件。

文件在 trunk file 内的 offset 编码到文件名，决定了其在 trunk file 内的位置是不能更改的，也就不能通过 compact 的方式回收 trunk file 内删除文件的空间。但当 trunk file 内有文件删除时，其删除的空间是可以被复用的，比如一个100KB 的文件被删除，接下来存储一个99KB 的文件就可以直接复用这片删除的存储空间。

HTTP 访问支持

FastDFS 的 tracker 和 storage 都内置了 http 协议的支持，客户端可以通过 http 协议来下载文件，tracker 在接收到请求时，通过 http 的 redirect 机制将请求重定向至文件所在的 storage 上；除了内置的 http 协议外，FastDFS 还提供了通过 apache 或 nginx 扩展模块下载文件的支持。



其他特性

FastDFS 提供了设置/获取文件扩展属性的接口（setmeta/getmeta），扩展属性以 key-value 对的方式存储在 storage 上的同名文件（拥有特殊的前缀或后缀），比如/group/M00/00/01/some_file 为原始文件，则该文件的扩展属性存储在/group/M00/00/01/.some_file.meta 文件（真实情况不一定是这样，但机制类似），这样根据文件名就能定位到存储扩展属性的文件。

以上两个接口作者不建议使用，额外的 meta 文件会进一步“放大”海量小文件存储问题，同时由于 meta 非常小，其存储空间利用率也不高，比如100bytes 的 meta 文件也需要占用4K（block_size）的存储空间。

FastDFS 还提供 appender file 的支持，通过 upload_appender_file 接口存储，appender file 允许在创建后，对该文件进行 append 操作。实际上，appender file 与普通文件的存储方式是相同的，不同的是，appender file 不能被合并存储到 trunk file。

问题讨论

从 FastDFS 的整个设计看，基本上都已简单为原则。比如以机器为单位备份数据，简化了 tracker 的管理工作；storage 直接借助本地文件系统原样存储文件，简化了 storage 的管理工作；文件写单份到 storage 即为成功、然后后台同步，简化了写文件流程。但简单的方案能解决的问题通常也有限，FastDFS 目前尚存在如下问题（欢迎探讨）。

数据安全性

- 写一份即成功：从源 storage 写完文件至同步到组内其他 storage 的时间窗口内，一旦源 storage 出现故障，就可能导致用户数据丢失，而数据的丢失对存储系统来说通常是不可接受的。
- 缺乏自动化恢复机制：当 storage 的某块磁盘故障时，只能换存磁盘，然后手动恢复数据；由于按机器备份，似乎也不可能有自动化恢复机制，除非有预先准备好的热备磁盘，缺乏自动化恢复机制会增加系统运维工作。

- 数据恢复效率低：恢复数据时，只能从 group 内其他的 storage 读取，同时由于小文件的访问效率本身较低，按文件恢复的效率也会很低，低的恢复效率也就意味着数据处于不安全状态的时间更长。
- 缺乏多机房容灾支持：目前要做多机房容灾，只能额外做工具来将数据同步到备份的集群，无自动化机制。

存储空间利用率

- 单机存储的文件数受限于 inode 数量
- 每个文件对应一个 storage 本地文件系统的文件，平均每个文件会存在block_size/2的存储空间浪费。
- 文件合并存储能有效解决上述两个问题，但由于合并存储没有空间回收机制，删除文件的空间不保证一定能复用，也存在空间浪费的问题

负载均衡

- group 机制本身可用来做负载均衡，但这只是一种静态的负载均衡机制，需要预先知道应用的访问特性；同时 group 机制也导致不可能在 group 之间迁移数据来做动态负载均衡。

原文

http://blog.yunnotes.net/index.php/fastdfs_design/?utm_source=tuicool

你是一个努力工作的程序员吗？还是一个懒惰的程序员？

当一个人在完成一件体力工作时，你很容易评估他是否在努力的工作。你可以观察他的物理动作，看他流了多少汗水。你还可以看到他工作的成功：砖墙在砌高，地面上挖的坑在变大。对努力工作的认可和褒奖是人性中非常基本的本能反应。这也正是为什么人们对体力耐力体育活动如此着迷的原因之一。这种对体力上的辛苦工作的本能的赏识，在遇到管理一群技术创造型的员工时，却成了一个麻烦问题。高效的脑力工作者通常会被看作并没有在努力的工作。

早在2004年，我还是一个初级程序员，工作在一家有线电视公司，在一个大型团队中开发财务和供销系统。跟所有的大型系统一样，这个系统由很多的相对独立的模块组成，分别由一些个人或小团队负责。其中模拟电视和数字电视的财务和供销系统几乎完全独立，分别由两个团队开发。

模拟电视开发组决定在早期的微软 Biztalk 平台上开发他们的系统。由这个公司的4个小伙和微软的一个团队共同开发，并负责产品环境的运行。他们看起来真的工作的十分辛苦和努力。你经常能看到他们加班到深夜或周末加班。每个人都会随时放下手中的活儿来解决正式环境中突现的问题，经常会在一张桌子前一群人围绕着

一个小伙，各自说出自己的见解，讨论什么地方错了，应该如何修正。工作气氛永远是热火朝天，每个人都能看到这些——即使只是经过瞟一眼，不仅仅从整个团队讲，而是他们每个人都真的真的工作的很努力。

数字电视供销系统开发团队却是完全的不同。代码几乎是由一个家伙写的，我们就叫他大卫吧。我是一个初级程序员，在团队里做维护工作。起初我在理解他的代码时遇到了很大的麻烦。他的代码里没有很长的过程，通常我的代码会把很多操作放到一起，相反，他的代码里有大量的很小的类文件和只有几行代码的小方法。好几个同事都抱怨大卫把代码搞的过度复杂了。但大卫耐心教导我，建议我去读几本面向对象编程的书籍。他给我讲设计模式，SOLID 编程原则，单元测试等知识。很快，我对他的代码开始有了理解，我越研究他的代码，越欣赏这些程序中优雅的设计。这些代码放到产品环境中非常好用，运行稳定的干着它们的工作。这些代码修改起来也相当简单，因此，一些新功能的增加变得轻松容易。单元测试保证了大部分的 bug 都阻挡到了正式环境之外。

这些做法产生的结果就是，我们看起来完全不是在十分努力的工作。我们5点半准时下班，周末从来没有加过班，我们从来没有发生过一大群人围绕着一个人数小时的讨论正式环境中的错误是怎么发生的场景。在外人看来，我们肯定是被分配了一件相对容易的任务。但事实上，需求都是十分相似的，我们只是更好的设计和实现了这个系统，有更好的支持系统基础架构，特别是单元测试。

管理部门宣称他们要根据员工的工作表现涨薪。当轮到老板跟我谈话时，老板说只给那些工作真的努力的员工涨工资才显的公平。而我们的团队看起来对公司发展的好坏并不太在意——跟那些放弃了自己的晚上和周末的英雄们相比。

这家公司是一个稀有的实验室，你可以将好的软件设计和坏的软件设计、好的团队特征和不好的团队特征的影响效果做一个直接的对比观察。大多数的公司里不可能提供这种比较的机会。你很难说这些挥汗如雨、工作到深夜和周末、坚持冲在灭火第一线的小伙们是为了开发一个真的非常非常复杂的系统而展示了伟大的付出，还是就是一次失败。除非你有能力提供两个团队来竞争，让他们解决同样的问题，可是哪个公司愿意做这样的事情呢。相反，如何看待那些坐在角落里，朝九晚五，看着像是整天上网读什么东西的程序员呢？是他们善于写出强健稳定的代码吗？还是分配的活儿比其他人容易？在常人的眼里，前一个团队的小伙们是在努力的工作，而第二个不是。努力工作值得赞扬，懒惰可耻，不是吗？

我敢断言，表面上看起来工作很努力通常会是一种失败的信号。在高压下，在一个不断被打搅的环境中，软件开发通常是不能干好的。长时间的工作往往不是一个好的方式。有时解决一个难题的最好的方法是停止思考，出去散散步，或更好的，去睡一个好觉，让潜意识帮你解决。我最喜欢的一本书就是20世纪英国数学界领军人物 G. H. Hardy 先生写的《A Mathematician’s Apology》。在这本书里，Hardy 先生描述他的日常规律：上午4小时的工作，下午看板球比赛。他说一天超过四小时的高强度脑力劳动都是无意义的，也是无效率的。

对于那些管理者们，我想说的是，判断一个人要看结果，要看开发出的软件的好用与否，而不是看他们表现的是如何在努力的工作。很反直觉吧，你其实最好不要坐在这些程序员中间，这样能保证你不受传统的、本能上的评判指标的影响，这样你才能对他们的产出有更好的认识。远程工作是特别有效的一种做法，你只能通常他们的产出来评判他们，而不是省事的观察他们是否8小时都坐在办公桌前对着 IDE 噼里啪啦的敲着键盘或“热心的”围聚在另外一个人的桌前提供着“有效的”建议。

原文

http://www.aqee.net/are-your-programmers-working-hard-or-are-they-lazy/?utm_source=tuicool

各式各样的极品程序员，你属于哪一种

摘要：极品程序员，或许在你看来就是那些比较另类的程序员。但是这里所列举的极品程序员的类型不是你想象的那样。他们除了极品这一特征之外，还具备其他优点。本文详述了宠物专家型程序员、奥术师型程序员等等。

本文作者 Aaron 是 MarkedUp 创始人兼.Net 开发者。在这篇文章里，作者根据自己平时在工作中的所见所闻，列举了几个比较有代表性的极品程序员类型。他们各有自己的特点和性格，在软件开发工作中，这种特点和性格显露无疑。（以下为编译内容）

在日常工作里肯定会发现很多有趣的事情，极品程序员所做的事就是很有意思的。事实上，现在所讨论的极品程序员主要是从他们的判断力、行为举止、个人态度和匪夷所思的工作方式上来判断的，有的时候这些程序员一味的只是追求文档上的内容，而不擅于用分析方式来解决问题。

几乎每一个软件开发者多多少少都会出现头脑短路的现象，也就是说，下面所列举的各种极品程序员类型，总有一款是属于你的。

1. 宠物专家型程序员：独爱一种技术

这种类型的程序员所遇到的致命错误就是：只钟爱一种技术，对别的技术不来电。

其实这不是什么值得表扬的优点。因为这种程序员看上去就像是“天生注定爱上这种框架的程序员”，对于自己喜欢的技术可以说是放不开、丢不掉。甚至在生活里也是一个打破砂锅问到底的人。

不管问题是什么，他的回答总是跟他的性格类型脱不了关系：

- “嘿，我们需要在 Rails 框架里实现一个内容管理系统，但是我们应该用哪一个数据库呢？”Mongo
- “多用户博客引擎吗？”Mongo
- “关键业务一致性系统？”Mongo
- “库存管理系统？”Mongo
- “电子病历系统？”Mongo
- “分布式数据仓库？”Mongo

宠物专家类型的程序员在任何软件开发项目里都会找出各种各样的理由，也不管这些理由是否具有实际意义。但是，一旦你试图反驳他的观点或者是想法的时候，他们在感情上是无法接受的，即使你和他的关系特别好也是没有用的。如果他知道自己的想法错了，也不会告诉任何，而是更愿意在最后一秒的关键时刻偷偷的使用别人的正确方案。

2. 奥术师型程序员：保守主义者，不到万不得已不做修复

奥术师类型的程序员的目标是至高无上的：不考虑成本，尽一切可能保护系统的正常运行和系统的完整性。除此之外，这类程序员信奉一个很简单的哲理，这个简单的哲理也就是引领他们在软件开发和管理实践中的一个基本准则：只要还能将就使用，不到万不得已千万不要去修复。不管什么样的软件，只要是在奥术师的保护下，就一直是使用那个平台、那个部署程序、那个数据库还有那个永远不会改变的操作

系统。但是你不必担心，因为奥术师会一直盯着这些看上去老旧的组件，也许他不能赢得每一场战斗，但是他一定会为了每一场战斗而凶猛拼杀。

在奥术师的世界观里，过去也代表着未来。所以，他将会和那些想要升级他的1981年发布的的 PASCAL 代码库的人作战到底，哪怕是泪流满面。

3. 未来派程序员：赶潮人，追求最前沿的技术

未来派程序员可以说是奥术师程序员的对立面——今天就是未来，任何使用昨天的开发工具编写的代码在未来派眼里看来都是无与伦比的厌恶。他们的目标可不是什么哲理——追求最新最前沿的技术。

如果你看到未来派程序员甩着鼻涕在你面前吹嘘最近在 Hacker News 上看到关于 JavaScript 的最新消息而滔滔不绝的时候，你千万不要惊讶，因为他们没像是在 Justin Bieber 演唱会上那样在你面前尖叫就已经很照顾你了。即使有的时候在 Microsoft Research or the Server and Tools Team 发布新产品的时候，即使他们根本不理解那些产品的用途，也会显得很热情。

不过，需要提醒的是，如果你是一个未来派程序员，那么 DevOps 工程师，QA 工程师和 Release 工程师这些职位是未来派程序员的天敌。

4. 信息囤积者型程序员：小心谨慎，不愿公布代码

信息囤积者型程序员是一个很谨慎的人，但是对自身的谨慎行为充满不确定性。他们生活在一个认知失调的世界里：对自己的工作感到自豪，但是需要帮助的时候又不愿意让别人知道。

所以囤积者通常会隐藏代码，不愿意公布代码。小心翼翼地避免访问记录，更不愿意别人跟踪他所做的一些修改记录。他最大的恐惧就是遇到代码合并冲突，因为这样的暴露风险是最大的。

也许囤积者型的程序员很乐意告诉你他的工作是多么了不起，但是如果你想要知道他的代码估计会很难。最终，信息囤积者型的程序员注定要失败，这样的做事行为不被别人接受，存在也是没价值的。只希望在短期内能有较好的改善。

5. **艺术家型程序员：**质量？美观？不可兼得

艺术家型程序员简直就是囤积者和未来派程序员的表兄弟，艺术家型程序员会倾入所有的心思来构建完美的代码行。此外，艺术家型程序员也是一个易动感情的人——他所创造出的软件就是他感情的流露，也是他个人才华的生动化身。

艺术家型程序员还会考虑这样的问题：如果我使用的 JavaScript 里不添加分号的话，在语句上是不是更加漂亮呢？如果把这个块包装起来，是不是更加完美？他们最在乎的是美观，其次才是质量。

艺术家型程序员也不是这个行业所提倡的那种程序员，他们无法客观的讨论评价自己的作品，也无法定位自己在同事当中的位置。

6. 孤岛型程序员：性格孤僻者，代码就是一切

孤岛型程序员是整个程序员行列里最孤独的人，这样的程序员在软件创造方面有很多很大的欲望，但是他们的短板是不愿与人交往。孤岛型程序员的理想工作条件是与外界交流最好是保持在最低限度，而且严格控制在他方便的时候。他的生活里只是代码，没有人类。

但不幸的是，现实和理想往往是有很大的区别的，但是这些程序员必须为了生存而去公司谋求工作，于是被迫与同事或客户沟通，这对他们来说的确是一个巨大的负担。所以他们只能躲避——躲避会议、躲避电话沟通、关闭邮件接收器等等。他们遇到问题的时候宁愿查询上百件项目文档也不愿意问自己的队友。和信息囤积者型程序员一样，孤岛型程序员注定要失败。软件开发就像是一项团队运动，不接受那些不遵守规则的另类。

7. 敏捷型程序员：急功近利，常常半途而废

敏捷型程序员是一个功利主义者，致力于需求改善软件效率，以及个人和团队的生产力。但不幸的是，他对“敏捷”哲理的理解和实施策略实在是呆板和僵化，不禁使人发笑。

敏捷型程序员的初始意图是很高尚的：改善软件开发方式。这样的程序员做起事情来是比较雷厉风行的，但是也有缺点：任何讨论时间超过四小时的问题最后都会变成泡沫；任何在最后冲刺阶段所做的项目基本上都会缩水。

另外值得注意的是，大部分敏捷型程序员都有一个通病：自命不凡。在程序开发过程当中，尤其是在选用一些小的组件的时候，坚持己见，导致网络文件系统出现错误或者是驱动程序实现效果不理想。

8. 文盲型程序员：编程入门者，功底浅薄

顾名思义，文盲型程序员在阅读别人的源代码的时候总是会遇到很多麻烦，有的时候基本上是看不懂别人的代码。

换句话说，文盲型程序员和孤岛型程序员像是一对表兄弟，只对他们钟爱的编程语言花时间去理解基本的编程结构，全面掌握编程语法，但是看到其他程序员所写的代码的时候，完全是一窍不通。在这种情况下，我们称之为“code-blind”程序员。

当面对其他开发者问“你为什么不使用我们标准的接口来生成一个对话”的时候，文盲型程序员通常是盯着自己的脚尖然后是喃喃自语。

以上只是列举了一部分类型的极品程序员案例，当然，极品并不代表不好，只要把这种极品特征合理应用，它就会成为你的特长。不知道你是不是一个极品程序员？

原文
http://www.csdn.net/article/2013-12-24/2817904-The-Taxonomy-of-Terrible-Programmers?utm_source=tuicool

告别码农，成为真正的程序员

本文是我借助 Google 从网上拼凑的文章，可能条理不是很清晰，希望对广大程序员们有些帮助。

一、成长的寓言：做一棵永远成长的苹果树

一棵苹果树，终于结果了。
第一年，它结了10个苹果，9个被拿走，自己得到1个。
对此，苹果树愤愤不平，于是自断经脉，拒绝成长。
第二年，它结了5个苹果，4个被拿走，自己得到1个。
「哈哈，去年我得到了10%，今年得到20%！翻了一番」。
这棵苹果树心理平衡了。
但是，它还可以这样：**继续成长**。
譬如，第二年，它结了100个果子，被拿走90个，自己得到10个。
很可能，它被拿走99个，自己得到1个。
但没关系，它还可以继续成长，第三年结1000个果子……
其实，得到多少果子不是最重要的。
最重要的是，苹果树在成长！
等苹果树长成参天大树的时候，那些曾阻碍它成长的力量都会微弱到可以忽略。
真的，**不要太在乎果子，成长是最重要的**。
切记：
如果你是一个打工族，遇到了不懂管理、野蛮管理或错误管理的上司或企业文化，那么，提醒自己一下，**千万不要因为激愤和满腹牢骚而自断经脉**。
不论遇到什么事情，都要做一棵永远成长的苹果树，因为你的成长永远比每个月拿多少钱重要。

二、人人都需要时间管理

一项国际查表明：**一个效率糟糕的人与一个高效的人工作效率相差可达10倍以上**。
哈佛有一个著名的理论：**人的差别在于业余时间，而一个人的命运决定于晚上8点到10点之间**。
每晚抽出2个小时的时间用来阅读、进修、思考或参加有意的演讲、讨论，你会发现，
你的人生正在发生改变，坚持数年之后，成功会向你招手。
我曾整理了一份『免费的编程中文书籍索引』(去 github 查看,也可以到 CSDN CODE)，每天抽出半个小时来读一读。
时间管理可以帮助您把每一天、每一周甚至每个月的时间进行有效的合理安排。
运用这些时间管理技巧帮您统筹时间，对于每个人来说都是非常重要的。
在时间管理中，**计划组织**相对于其他技巧来说是最简单的一种。
比如，所有的时间管理建议都包括在一些表格当中，在表格中把您想要完成的任务填进去。
对很多人来说，这是最简单和普通的了。

三、别人能成功的事，未必自己就能成功

飞机上，乌鸦对乘务员说：给爷来杯水！
猪听后也学道：给爷也来杯水！
乘务员把猪和乌鸦扔出机舱，乌鸦笑着对猪说：傻了吧？爷会飞！

外界因素是一种约束条件，自身能力也是一种约束条件，往往更重要。
所以，别人能成功的事，未必自己就能成功。
四、你搜索到的只是网页，不是知识

知识的类型及它在程序员大脑中如何成长。
有三类知识：
概念知识（为什么、是什么、如果——语义上的）——理解软件系统构建过程中的概念、原理、关系及主要方法。
实践性知识（如何做——过程中的）—— 关于如何解决特定编程问题的知识。
这类知识不需要深入理解实现方法选择过程中隐含的概念及基本原理。
隐性知识（专业知识、经验及直觉）——基于软件系统实现过程中所积累的个人经验，在大脑中形成的内在知识。
这类知识很难传授，因为它的大部分都存储在我们的潜意识中。
可解决实际问题的高效搜索
A. **查找 Seek**
定义 Definition——弄清楚要解决什么问题，并以要查找的内容为焦点。
检索 Retrieval（使用标准的 Google、代码搜索或其他的检索引擎）—有很多关于如何高效的使用检索引擎的建议。
浏览结果 Browse（内容的质量、可信度及专业技术的水平；如果资料的可信度过低，无须再看）-> 阅读 -> 评估（人力物力、所需工具及函数库）
B. **使用 Use**
复制代码 - 单独复制（针对这一目的，带有长钉技术的显式单元测试最适合）。
清除代码 - 仅保留最小限度、相关性代码，清除解决方案中的其它代码。
应用代码 - 在系统中应用代码。
C. **学习 Learn**
理解 Understand——你做了什么及你为什么那样做——从代码和实现中学习。
扩充知识 Expand——

- 实践性知识 Practical：解决问题的特定方法、技巧及风格；
- 概念知识 Concept：学习新概念、提炼现有的并构建自己的概念；
- 隐性知识 Recessive：明智地使用并学习搜索到的解决方案，经验会自然而然地得到增长。

收集 Collect（链接、意见、参考文献、阅读清单）—任何对你今后搜索、发现及学习有用的有趣信息。为这些目标积累知识。
还有一点也很重要：**分享与交流**。
最后还是我在博客中经常写道的那句话（不要嫌我罗嗦，再写一遍），**学历代表过去，能力代表现在，学习能力代表未来**。

原文 http://justjavac.iteye.com/blog/1995172?utm_source=tuicool

陋谈创业早期资金积累学习

当大家看到这篇文章时，我还在北京，打算周六回常州，然后下周马不停蹄的

赶到上海。

这次来见了一些客户和朋友，收获颇大，感触颇大，和大家一起分享。

九年前来北京北漂时认识了很多北京本地的朋友，说实话，北方人对待朋友的感觉是南方人不可比拟的，当然两者各有千秋，我只能说我个人角度更愿意和北方人做朋友。纯个人看法。

朋友 A 当时见到我的第一个动作是发烟。

“来根中南海吧“，朋友 A 很热情。当时我在老家一般都抽20左右一包的香烟（小城市人特爱面子，收入不高，但是香烟抽的很高级。另，那是在2004年）

我抽了，第一次抽中南海感觉是很高档的香烟，后来询问价格发现只要□4.5。

再后来得知，该朋友 A 在某一家事业单位工作，每个月收入是2000多一点。由于是本地人所以压力不是特别特别大。作为程序员的我很有感触，当时我的收入是5000（不要和现在比哦），很有优越感。

九年后，也就是现在我再次来到北京。朋友 A 已经辞职了，自己开了多家公司。我惊讶，我努力写程序10年，最多也就开一家公司，其实这几年我一直被资金所困，每当有个机会或者项目时总是没钱，贷款？没东西抵押怎么贷款？

我很虔诚的向朋友请教生存之道，于是对话开始。

“你的事业咋弄出来的呢？“，我虔诚的很，表情很严肃，防止朋友 A 认为我是想问他借钱。

“搞项目啊，顺便做一些投资”，朋友 A 不屑一顾。

“咋弄的钱呢？难道你去拿到风投了“，我问的很俗，在我眼里只有靠风投才能有钱。作为程序员的我认为这是唯一一条道路了。

“没有“，朋友 A 否认。

“求给真经“，我已经不耐烦了。

“房子嘛”，朋友 A 眼神很“鼻屎“。

20分钟后，具备超强理解能力的我终于知道运作方式了。

1。朋友 A 当年咬牙买了一套房子，一切都为了结婚，求懂。没房子肯跟着你的 MM 在这个年代不靠谱。

2、几年后北京的房价长了 N 倍，注意 N 这个字。

3、朋友 A 没有卖这套房子（卖掉是很多人的想法，实际上这是错的）

4、朋友 A 吧该房子到银行做了评估后，质押了出去。注意：假设他买的时候是6000一平，现在市值已经达到了3万一平，质押的话应该是按市值再打个折扣。

5、关于房子质押的概念和形式，请大家自行查找度娘学习。

6、于是朋友 A 有钱了。不过该房子的使用权还是他的，于是他把房子又租了出去。

7、于是他买了一辆车，到远一点的地方又租了一套房子，用他原来的房子收的租金还自己租的房子。差价还能生活。

8、无限期付利息。反正房子理论上还是你的，只不过手上有钱，这时可以拿出来注册个公司。

结论是：流动资金有了，公司注册资金也有了，住的地方也有了，反正自己没花钱。就算不干啥，依然有收入。（请大家结合前面讲到过的理财等知识联想）。

我的观点：

1、创业靠啥：资金、努力、人脉和关系。其实其他还有几个要素，不过前面四个是最重要的。

2、要有魄力，要有市场敏锐感。这就是为何我一再强调，程序员不要闷着头搞技术。技术不能转化为利润都是扯淡，

3、AK47之父去了，前天我还看了中国近代史手枪的发展历史。开发手枪也是技术，只不过当时几个手枪之父除了技术能力很强的情况下，他们的商业头脑譬如和军方合作、抓住二次世界大战的时机都是很强的。

4、做技术是作技术，咱应该想着几年后怎么把技术变成白花花的银子。我们在媒体上看到的那些“那些只为技术付出一生，完全不为赚钱的例子”那是统治者给大家洗脑，希望大家好好打工不要有其他想法而 杜撰出来的。人家万恶的资本主义就很直接，搞这么多科研和技术，目的只有一个：利益。

5、商业头脑怎么培养？观察力、联想能力、学习精神、生活的压力。有这四个力你必定有商业头脑，不用学和自我培养。

6、要赚钱必须首先成为钱的主人，而不是钱的奴隶。慢慢的大家就会懂这句话到底是啥意思。

原文 http://www.shenyisyn.org/2013/12/27/4861.htm?utm_source=tuicool

新兴独立开发者的 PR 策略之语言与情感

当我们创建 Mode 7时，PR 策略的理念还有点傻：没人关心独立开发者以及他们所说的，所以 PR 策略的目标只是不断地高声喊出任何可能的内容。

这是一个全新时代：独立开发者不断接受访问，并被要求对当下的游戏新闻故事做出评价；这是件好事！在更大的公司中，我们通常都不存在 PR 过滤器和限制，所以我们能够更多地说出自己的想法，并让人们关注我们的游戏。

在本篇文章中，我主要要谈论以下内容：

关于做出声明与进行访问的想法

发行游戏内容

一般哲学内容

谈论某些内容

我们中的很多人都认识新闻记者，并且私下很喜欢与他们谈论一些内容；这是受到媒体的尊重，并且保持一切顺畅发展的主要元素——更别说基本的个人利益。但是有时候，妥善地选择朋友也很重要。

我认为，我们有时候总是很容易掉进一个聊天模式中，并忘记祸从嘴出。为了进行说明，我将以虚构的 GameHerbert 主机和 GameHerbert 产业为例。

“等等等等，独立开发应该这么说，不是吗？”或者

“每个人都知道这个：闭嘴”

在我开始前要先明确两个要点：

1）在此我并不是要告诉任何人我认为他们应该说什么，他们应该谈论什么主题等等。你可以做你想做的事，你当然需要表现出自己的个性。我想要做的是当某些事情发生在公众面前时哪些反应是我们所期待的。

2）我所掌握的一点是，人们在理解这些过程的直觉性会出现很大的变化。对于某些人而言，我所说的一切都是再明显不过的，但对于其他人而言，这可能是他们第一次仔细思考这些内容。也许你正致力于一个项目中，但却从未发布过任何有关这一项目的信息或接受过反问：那么本篇文章将对你很有帮助。

你所说的任何富有争议的内容都将成为前景，即使你只是将其当成悄悄话，并且这一内容也非与你有着直接关系。

Frozen Endzone(from pcgamer)

让我们假设我接受了一次有关《Frozen Endzone》的采访，在最后我说道：“它永远都不会出现在 GameHerbert 上：这是一个很难瞄准的平台，所以我们可能没有足够的时间耗在上面”

如果说我不是期待着看到“独立开发者说很难面向 GameHerbert!”的大标题，那只能说我误解了媒体。

我可能会抱怨道：“他们将整个采访全部集中在我最后说的那句话上!”但事实上他们有什么理由不这么做的？为什么我要期待着他们别这么做？

显然新闻记者始终都在想办法吸引别人去阅读自己写的采访内容。实际上，任何独立开发者（除了 Notch 或像 Jonathan Blow 这样的人）很难凭借自身因素成为新闻：你所说的才是访问中唯一的吸引力。如果你说了一些有关 GameHerbert 的有趣内容，这将会比你的作品更加受欢迎，并更有可能出现在头版上。

为什么媒体这么热衷做这些事？主要是因为资深的开发者会因为上述原因而抵触对于任何与自己的游戏无关的内容做出评论。

并且，你也可以坚持“不评论”直到问题结束。你可能会觉得在一开始这么做很愚蠢，但你真的可以这么做。我便听过有人说：“他们一直问我问题，所以最后我觉得自己应该说些什么才行。”

当然，我并不是说独立开发者应该拒绝做出更多“评论”。我只是说如果他们想这么做的话也是可以的。

任何具有争议的要点的说明或调整都将被删除或埋葬

这很重要——但总是被遗忘

以下是一个虚构的访问摘录：

“GameHerbert 是一台出色主机：带有嵌入式 USB 风扇，发光的猴子控制以及可伸缩的轮子。但我的意思是，它看起来就像一块沾了粪便的钻块，对吧？（每个人听了都会笑）。话虽如此，有时候我将 GameHerbert 放在包里只是因为我想要在上班路上玩《Super Rowing Boat Madness GT》，所以这真的是一个带有很多面的可靠产品。说实话，我认为它看起来不错。”

我只是提供了一个引用:“GameHerbert 看起来就像是一块沾了粪便的钻块。”这是一个很棒的引用，但如果我并不希望它成为新闻上的醒目引文的话，我便不知道自己到底是怎么了。这个世界中的任何新闻记者，不管是否友善，都将争抢这个话题。不过他们也有可能不会从“话虽如此”这句开始引用。他们也许会在正文中添加有关 GameHerbert 是一台优秀主机的内容，

这么做只是为了表明你是在适当的环境下说出这一内容。

1）醒目引文是没有环境的

2）没人在乎意外事件的前言或附言

我并不是在批评新闻记者：这是新闻的运转方式，这也是我们需要接受的方式。我们需要看到简短，易消化，古怪或有趣的点才会去阅读某些内容。

我也不是在批评易做出突发事件的开发者：就像我之后会说道的，有时候需要发生些事情让某一问题更加鲜明，从而帮助更广泛的社区进行理解。同样的，我们有时候也需要一个突发事件将问题带到最前线；有些人需要成为讨论这些问题的先驱，并对此做出批评。而他们从中获得的利益有时候还会超过其它结果。

丢下炸弹

当我第一次在公众面前讨论游戏时，我是与杰出的 Jason Wonacott 一起参加了一个座谈小组。他提供给我一个有关公共演讲的建议；我认为这也能应用于采访中。这一建议如下：

“思考你想要说的内容，然后想出一种有趣的方式去传达这一内容。避免过长。在开始说话时，稍微停顿下，然后开始说话，并适时停止。这是你能够多方面受用的方法。”

你甚至不需要做太多事：只要事先想出传达自己想说的内容的有效方法便很有帮助了——不管是对于你自己还是新闻记者。

信息控制？

现在我们所面对的文化是，来自独立开发者口中的“内容”越来越多了。

我们处于一个非常拥挤的派别中，为了让别人注意到自己，我们就需要讲更多话。再一次，这的确蛮棒的：我想要听到那些创造着我喜欢的内容的人说更多事。有时候这真的很有帮助；这不仅能够让许多开发者收益，同时也能够带给普通用户更多好处。

在 Twitter 上我们能做的最重要的一件事便是刊登许多新闻；不管你刊登的内容无不无聊都不重要。在 Facebook 上也是如此。我阅读过著名的 Destiny 关于自己如何吸引用户的博文，而这都是一些再平常不过的内容。

个性也很重要：人们之所以会频繁引用 Jon Blow 或 Phil Fish 的话语是因为他们所说的内容都具有很强的煽动性。这对于他们也有很大的积极影

响：对于他们来说出名也不是什么坏事。

临界物质

有时候，个人体验将帮助你更好地理解事情的发展。

去年，因为沉迷于改变游戏故事结局的理念，我在有限的时间内改变了《Frozen Synapse》的结局。我在 Twitter 上提到了这点，有些人鼓励我这么做，所以我便继续。最终这一行动引来了大量的关注。

其背后的灵感在于游戏粉丝强烈抗议《质量效应3》的结局，但我并不想评价这种情况的对与错。

我的想法是：

粉丝与游戏的关系已经结束了，特别是在很少人会去看结局的游戏中。

创造者的想法是去破坏他自己的作品；而我在自己并不满意的结局中也进行了效仿。

《Frozen Synapse》的结局没有多少评论——而我之前在编写这一结局的时候还觉得它很有煽动性。

我遭遇了来自《质量效应》粉丝接二连三的怒火。我很难理解他们的论点，但他们主要都是围绕着我是强烈支持或强烈谴责 Bioware 的想法展开。我快速统计了“支持”和“反对”Bioware 回应的比例，结果大概是50：50。

最近 Ian 告诉我，我犯了一个“巨大的错误”；我不知道从一个备受关注的问题中获取灵感会让人们认为我是在对此进行道德评判。

我从未玩过《质量效应3》，也不知道有关结局改变的具体情况，除了谣传的那些：可以看出我真的对此并不感兴趣。

在人生中我第一次收到了个人攻击信件，对方的言语非常激进；他们想要搞垮我;他们会侮辱我所写的内容;他们告诉我这代表憎恨我自己的用户;他们骂我是“白痴”。

我想从这次的经历中自己真的学到了许多：

情感优先

当我变得更加成熟时，我认为如何交流的最重要元素是我们怎么煽动别人的情感。最近我在 Twitter 上与 Nicholas Lovell 谈论了我们共有的特征：我们会非常强烈且武断地讨论一个要点并刺激反论观点;如果反论观点是正

确的，我们便会仔细思考片刻，并在得知自己错误之后回头。

这在某种情况下是可行的，但通常都算是一种糟糕的交流；这么做只会赶走别人。同样地，当你证明自己的观点是错误时你也可以长久坚持这个观点——这只是一种糟糕的行为罢了。我总是想搞明白为什么人们在争吵的时候会如此容易生气，产生消极情绪或进行自卫——“如果他们是对的，他们便能够证明我是错的；那为什么他们还要低落呢——难道他们不相信自己吗？”

持有正确观点或足够聪明通常都不是传达你自己的想法的重要元素。

从个人偏好出发

我改变的结局让一些人生气；所以他们立刻将愤怒的矛头指向我；他们尝试着说一些攻击我的话。很幸运的是没有一条评论真正让我感到失落；我可以想象处于相同情境下，即被一些消极的关注置于一种极其糟糕的状态下的人会是怎样的。这也是我们为什么能在网上看到这么多疯狂的个人敌意的原因：某些事情会触发人们的愤怒按键，从而将其变成攻击模式。因为情感优先因素，他们并不会去攻击内容：他们很少会真正了解到内容。他们会攻击人是因为这是避免接触论点的最直接方式。大多数人都害怕出错；而这能帮助他们避免这种担忧。

不存在情境

所有能够让我感到惊讶的事物都是那些缺少情境感知的事物。没人在最初事物的情境中看到新的结局；没人在最初事物的情境下告诉我新的结局；别人只是让我对某些内容做出评价，而这种解释必定是独家的；我猜作者已经过世了。

为什么我要一直讲这些？主要原因是：

控制并不能代表一切

我并不是真的觉得这是一种错误；当我们接收到负面关注时，我想大多数人都知道这是不合理的，并且是关于一个宏大计划中一个并不重要的主题（我的意思是，在任何人开始评论前，“叙述”并不等于《质量效应3》!）。如果这是关于具有社会煽动性的问题，那么直到今天我可能仍会收到来自网络的愤怒攻击，尽管我并未对任何事情做出明确的声明。这就是令人震惊的现实。

如果过多地受控于“信息”，你便会停止做任何事。当我们开始运行 Mode 7时，我们的政策便是可以“说任何事”；之后，当我们意识到自己的言语的影响力时，我们才开始有所收敛。当你的一条 tweet 出现在 Eurogamer（游

戏邦注：一个位于英国布莱顿的网站，其以电子游戏的新闻、评论、预展和采访为主）的头条，你便会开始专注于按压 Tweetdeck 上的蓝色小按键。不过我认为我们似乎走过头了；对于如何进入对话，我似乎担心太多。

所以我想看看，当在面对早前所引起的一些问题时，我们仍否变得更加开明。在公共场合讲话的进化是你如何与其他人交流的进化的一个子集；我认为这种进化是一辈子的过程，也是所有人都应该重视的过程。

有时候，所有人都会做错：我敢保证自己有时候真的也是如此。这是常有的事：如果你对于别人的回应报以理智的期待，那么你传达自己想法的能力也将有所提高，并且不会担心做错事。

原文 http://gamerboom.com/archives/80258?utm_source=tuicool#